Accepted Manuscript

A New Cell Space Devs Specification: Reviewing the Parallel Devs Formalism Seeking Fast Cell Space Simulations

Fahad A. Shiginah, Bernard P. Zeigler

PII: S1569-190X(11)00014-1
DOI: 10.1016/j.simpat.2011.01.006
Reference: SIMPAT 1055

To appear in: *Simulation Modeling Practices and Theory*

Received Date: 1 May 2010
Revised Date: 21 December 2010
Accepted Date: 24 January 2011

Please cite this article as: F.A. Shiginah, B.P. Zeigler, A New Cell Space Devs Specification: Reviewing the Parallel Devs Formalism Seeking Fast Cell Space Simulations, *Simulation Modeling Practices and Theory* (2011), doi: 10.1016/j.simpat.2011.01.006

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

A NEW CELL SPACE DEVS SPECIFICATION: REVIEWING THE PARALLEL DEVS FORMALISM SEEKING FAST CELL SPACE SIMULATIONS

Fahad A. Shiginah

Department of Electrical & Computer Engineering; Sultan Qaboos University. shiginah@squ.edu.om

Bernard P. Zeigler

Arizona Center for Integrative Modeling and Simulation. Department of Electrical & Computer Engineering; University of Arizona. zeigler@ece.arizona.edu

Abstract

This paper introduces a new specification for cellular DEVS models that assures high performance. It starts with the parallel DEVS specification and derives a high performance cellular DEVS layer using the property of closure under coupling. This is done through converting the parallel DEVS into its equivalent non-modular form which involves computational and communication overhead tradeoffs. The new specification layer, in contrast to multi-component DEVS, is identical to the modular parallel DEVS in the sense of state trajectories which are updated according to the modular message passing methodology. The equivalency of the two forms is verified using simulation methods. Once the equivalency has been ensured, analysis of the models becomes a decisive factor in employing modularity in cellular DEVS models. Non-modular models guarantee the efficiency of the models in contrast to the current cellular DEVS implementation approaches. This was achieved by converting the cell space partially or fully into atomic model in order to eliminate inter-cell messages. However, the new specification needs an automated way to implement and verify models since they might become complicated ones.

Keywords: DEVS, Cellular DEVS, Cell space Modeling, non-modular DEVS.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

1. INTRODUCTION

The cell space modeling approach divides the spatial space into discrete cells where local computations held at each cell are based on its own as well as its neighbors' states. In conventional DEVS implementation of cellular models (e.g. [1]), a cell is implemented as a DEVS modular atomic or coupled model. When detailed modeling of spatial dynamics is required, large number of cells are typically employed. This results in a large number of atomic models that communicate through message passing to carry out the global simulation. Therefore, the task of implementing large scale cellular spaces with highly active cells in DEVS will face the burden of huge number 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 [2-4] 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 [5, 6] 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.

Most research work in DEVS cellular space modeling has treated each cell as an atomic or coupled model and then either sought to speed up the simulation engine or introduce quantization to the model in order to reduce messages and transitions with attendant error. The work presented here applies enhancement methods to the model development level. This new error-free (i.e. quantization not employed) approach is designed to reduce the number of messages by encapsulating and transforming a group of cells into non-modular form. Instead of treating a single cell as an atomic DEVS model, the encapsulation method will group a number of cells in one atomic model. The resulting model will be a non-hierarchal, non-modular cell representation that gives a significant speed up which in conjunction with the simulator enhancements expected to give high performance on large cell spaces.

2. RELATED WORK

There are few related works that touched the area of converting coupled DEVS models into atomic models like [7] and [8]. Their implementation of the approach is by allowing the conversion during the compilation process. On the other hand, this paper applies the conversion into the specification and development process. The first work involved converting classical, rather than parallel DEVS, models and it did not target the cellular space in particular which made that approach a good speedup way for small size models. The other work also tried to convert large models into atomic ones to easily deal with them in Dymola. A conclusion was reached that there is no advantage of following the conversion approach since the overhead of handling large model is much greater than the messages overhead. This paper disagrees with that conclusion which can be considered an environment specific conclusion. Our initial work in [9] proved the significance of the approach for large cellular DEVS models.

3. BACKGROUND

3.1. Parallel DEVS Formalism (P-DEVS)

Discrete EVent System Specification (DEVS)[6] supports object orientation over modeling environments. Its theory provides a mathematical formalism for representing dynamic systems. The DEVS formalism was revised in

ACCEPTED MANUSCRIPT

[10] to reduce sequential processing and enable full parallel executions. The resulting parallel DEVS has the basic atomic model defined as:

$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$,

where

X is a set of input values (set of ports/values in coupled structures)

S is a set of states (set of ports/values in coupled structures)

Y is a set of output values

δ_{int}: $S \to S$ is the *internal transition* function

δ_{ext}: $Q \times X^b \to S$ is the *external transition* function

$Q = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$ is the *total state* set

e is the *time elapsed* since last transition

X^b denotes the collection of bags over X

δ_{con}: $S \times X^b \to S$ is the *confluent transition* function

λ: $S \to Y^b$ is the output function

ta: $S \to R_{0 \to \infty}$ is the time advance function.

An atomic model M in parallel DEVS remains in a state $s \in S$ for $ta(s)$ amount of time 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, in case of internal and external transitions collide, δ_{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. [6]

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:

$CM = \left\langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \right\rangle$,

where

X is the set of input values (set of ports/values in coupled structures)

Y is the set of output values (set of ports/values in coupled structures)

D is the set of components

for each i in D: M_i is a component which is an atomic model $M_i = \left\langle X_i, S_i, Y_i, \delta_{\text{int}\,i}, \delta_{ext\,i}, \delta_{con\,i}, \lambda_i, ta_i \right\rangle$

for each i in $D \cup \{\text{self}\}$: I_i is the influencees of i, i is not in I_i

self is the coupled model itself CM which allow external inputs and outputs

for each j in I_i : $Z_{i,j}$ is the i to j output translation function (coupling)

ACCEPTED MANUSCRIPT

$Z_{self,j} : X_{self} \rightarrow X_j$

$Z_{i,self} : Y_i \rightarrow Y_{self}$

$Z_{i,j} : Y_i \rightarrow X_j$

3.2. Closure under Coupling of Parallel DEVS

Closure under coupling, in parallel DEVS, states that every coupled model (*CM)* has its own equivalent atomic model (*M*). Generally speaking, the coupled DEVS model can be treated as a black box with input as well as output ports (*X* and *Y*) that form the first terms of equivalency in the atomic and coupled models. The state set (*S*) of the resultant model will be the total state sets of all the atomic coupled models. In addition, the time advance *ta(s)* will be the minimum of all the internal atomic models.

3.3. Cellular Space Models in DEVS

Cellular Automata (CA), which was first introduced by John Von Neumann in the 1950s [11], has been widely used in simulating complex systems. The domain of application of CA includes fluid and mass flow, natural hazards, many other sorts of pattern recognition, image processing, ecosystems, and traffic modeling. In addition, it has been used as solutions for common computational needs like networking, solving differential equations, and distributed computing. [11-23]

The cellular automata applications, based on the discrete time simulation, consume the computation power in doing computations to update all cells in every single iteration. In a wide range of applications, there are a lot of cells that are not required to be updated at every step which makes the discrete time approach inefficient. In addition, the selection of the time step size has a significant impact on the simulation accuracy. High accuracy requires a very small step size which, in turn, requires huge computational resources. The discrete event approach overcomes these problems by dedicating computational resources to the cells that actually perform state transitions and hence avoiding unnecessary computation on inactive cells. Due to these advantages, many efforts were dedicated to employ the DEVS approach to cellular automata applications (e.g. [1, 4, 24, 25]).

The conventional cellular DEVS approaches divide the spatial space into discrete cells where local computations are done in each cell. A cell is implemented as an atomic DEVS model which performs the local computations internally based on its own state as well as the neighboring states that are received through the external ports. The cell space is implemented as a coupled DEVS model that contains a number of cells that are arranged in an array. The neighboring rule followed in a specific application determines the internal port couplings between cells and the boundary couplings that connect the cells at the borders with other cells in different cell spaces. Figure-1 illustrates the conventional 2-D cellular space implementation in the DEVS formalism.

ACCEPTED MANUSCRIPT

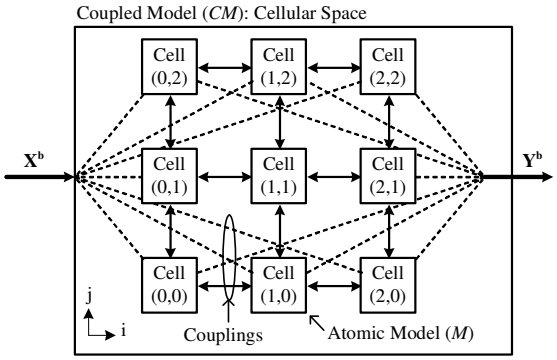

Figure-1: Cellular Space in DEVS

4. NEW FRAMEWORK FOR CELLULAR DEVS MODELING

In this section we formulate a new cellular space DEVS specification to achieve fast simulation execution. The basic idea is to convert the cell space entirely into DEVS atomic models. This process can be seen as converting modular cells into non-modular form inside the atomic cell space where each cell can access the state variables of its neighboring cells.

4.1. Converting Cell Space Model into Atomic DEVS

As a special case of the closure under coupling property in DEVS, cellular space models can take advantage of this property in gaining speedup by converting coupled models into atomic ones. In this approach, scalability will not be an issue since all cells (i.e. atomic models) have identical transition functions that will be applied to all cells iteratively. This implies that the model will not be required to search for the functions in a huge pool for the encapsulated atomic models. In addition, by fully composing the internal models into non-modular form, the inter-cell communication messages will be eliminated and result in simulation speedup.

4.2. Closure under Coupling of Parallel DEVS Applied to Cell Spaces

Closure under coupling of parallel DEVS states that a coupled model can be represented with its atomic P-DEVS equivalent which is defined as follows:

Given P-DEVS coupled model $\left\langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \right\rangle$, we define a basic atomic model $\left\langle X^a, S, Y^a, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, ta \right\rangle$ Where $M_i = \left\langle X_i, S_i, Y_i, \delta_{\text{int}i}, \delta_{exti}, \delta_{coni}, \lambda_i, ta_i \right\rangle$

for each $i \in D$, $X \equiv X^a$, $Y \equiv Y^a$, $S = \underset{d \in D}{\times} Q_d$, $ta(s) = minimum\{\sigma_d \mid d \in D\}$, $s \in S$, $s = (\ldots, (s_d, e_d), \ldots)$, $\sigma_d = ta(s_d) - e_d$, and the transition functions are defined as follows:

δ_{int}: $\underset{d \in D}{\times} Q_d \rightarrow \underset{d \in D}{\times} Q_d$

δ_{ext}: $\underset{d \in D}{\times} Q_d \times X \rightarrow \underset{d \in D}{\times} Q_d$

δ_{con}: $\underset{d \in D}{\times} Q_d \times X \rightarrow \underset{d \in D}{\times} Q_d$

ACCEPTED MANUSCRIPT

$\lambda: \underset{d \in D}{\times} Q_d \rightarrow Y$

In the case of cellular space models, all components $d \in D$ are P-DEVS atomic cells which are identical processing objects with the same transition functions as well as output functions δ^*_{int}, δ^*_{ext}, δ^*_{con} and λ^*. For this special case of the DEVS coupled model, the resultant overall cell space atomic functions will be as follows:

$\lambda(s)=\{ Z_{d,self}(\lambda^* (s_d))| \ d \in IMM(s) \wedge d \in I_{self}\}$

$\delta_{int}(s) = (\ldots, (s_d', e_d'), \ldots)$

$$\text{where } (s'_d, e'_d) = \begin{cases} (\delta^*_{\text{int}}(s_d), 0) & d \in INT(s) \\ (\delta^*_{ext}(s_d, e_d + ta(s), x^b_d), 0) & d \in EXT(s) \\ (\delta^*_{con}(s_d, x^b_d), 0) & d \in CONF(s) \\ (s_d, e_d + ta(s)) & otherwise \end{cases}$$

$\delta_{ext}(s,e,x^b) = (\ldots, (s_d', e_d'), \ldots)$

$$where\ 0<e<ta(s)\ and\ (s'_d, e'_d) = \begin{cases} (\delta^*_{ext}(s_d, e_d + e, x^b_d), 0) & self \in I_d \wedge x^b_d \neq \Phi \\ (s_d, e_d + e) & otherwise \end{cases}$$

$\delta_{con}(s,x^b) = (\ldots, (s_d', e_d'), \ldots)$

$$where\ (s'_d, e'_d) = \begin{cases} (\delta^*_{\text{int}}(s_d), 0) & d \in INT'(s) \\ (\delta^*_{ext}(s_d, e_d + ta(s), x^b_d), 0) & d \in EXT'(s) \\ (\delta^*_{con}(s_d, x^b_d), 0) & d \in CONF'(s) \\ (s_d, e_d + ta(s)) & otherwise \end{cases}$$

Given that the cell sets are defined as follows:

$IMM(s)= \{d \mid \sigma_d = ta(s)\}$	*Set of imminent cells*
$INF(s)=\{d \mid i \in I_d,\ i \in IMM(s) \wedge x^b_d \neq \emptyset\}$,	*Set of cells about to receive input,*
	where $x^b_d = \{Z_{i,d}(\lambda_i(s_i)) \mid i \in IMM(s) \cap I_d\}$
$CONF(s)=IMM(s) \cap INF(s)$	*Set of confluent cells*
$INT(s)=IMM(s)-INF(s)$	*Set of imminents those receiving no inputs*
$EXT(s)=INF(s)-IMM(s)$	*Set of non-imminents those receiving input*
$UN(s)=D-IMM(s)-EXT(s)$	*Other cells*

4.3. Event List Handling

The approach in [7] used the idea of closure under coupling to compose general (including non cell space) coupled models to gain some speedup. Unfortunately, those approaches do not guarantee speed up in large scale models. The first factor of scalable speedup in our approach is that it targets cell space models where all the cells are having the same transition functions. This property will ease the process of composing coupled cell spaces by moving these transition functions to the cell space level and implementing an iterative approach to apply these

ACCEPTED MANUSCRIPT

functions to the active cells. Since we are implementing the whole framework in the discrete event simulation domain, we need a discrete event list handler that manages, for the atomic cell space, the list of active cells at each simulation time, namely the cells to which the cell transition functions must be applied.

Introducing the event list obviates the requirement that each cell keeps track of its own timing since time management will be handled by scheduling events on the events list. The events list is employed at the level of the resultant atomic model which will contain the future events expected to happen for the cells. An event record will be in the form that describes when the event is scheduled to occur (next event time) and where it will happen (which cell). The elapsed time of each cell inside the DEVS coupled model is normally handled by the coordinator. However, since we are replacing the coupled model by its atomic model equivalent, it will be the atomic model's responsibility to keep records of the elapsed time for each of its internal cells. Therefore, the atomic model should keep a variable that store the current simulation time for the cell space block that it represents and a vector of cell's history times which store when a specific cell was accessed or had a transition time last. The elapsed time can be obtained by subtracting the history time of a specific cell from the current simulation time.

Within the DEVS framework, a cell can only receive an external message when the time advance has expired at another cell inside the same coupled model or an external message was received by the coupled model's input ports. Accordingly, the events list implementation just stores the time advances of the active cells and upon time advance expirations, the event list handler must add the neighboring cells of the imminent cells to a receiver group. This group will be the list of the cells that may receive external messages. In addition, when the coupled model receives external messages, the handler should identify the cells that should be aware of these new messages and add them to the scan list for external transitions. The events list handler is the responsibility of the resultant atomic model which, as a DEVS model, must implement all functionality solely using its transition functions.

Now, we have a cell space having an events list (£)with events $ev = (time, i)$, where

$ev \in £$, $time: R_{0\to\infty}$, and $i \in D$.

Therefore,

$$CellSpace=\left\langle X,Y,D,\{Cell_i\},\{I_i\},\{Z_{i,j}\},£\right\rangle \rightarrow \left\langle X,S,Y,\delta_{\text{int}},\delta_{ext},\delta_{con},\lambda,ta\right\rangle$$

where

$Cell_i = \left\langle X_i,S_i,Y_i\right\rangle$ *for each* $i \in D$,

$ta(s) = minimum\ \{time \mid (time, i) \in £\}$.

This means that in the new representation, the cell is no longer an active processing unit. It just stores the state variables with no timing involved at its level. It still has ports and messages which means it is not yet in non-modular form. The task of conversion to non-modular form will be done in the next subsection.

Now, the events list (£) will play role in defining cell groups as follows:

$IMM(s)=\{ j \mid (ta(s), j) \in £ \}$

Given that $X^b_j=\{Z_{i,j}(\lambda^*(s_i)) \mid i \in IMM \wedge i \in I_j\}$ *, the collected outputs of the imminents*

$INF=\{ j \mid (i \in I_j \mid ((ta(s),i) \in £) \wedge X^b_j \neq \varnothing)\}$, *receiving cells influencees*

ACCEPTED MANUSCRIPT

$INT = \{ j \mid ((ta(s), j) \in £ \wedge X^b_j = \varnothing)\}$,

$EXT = \{ j \mid (i \in I_j \mid (ta(s),i) \in £ \wedge (ta(s),j) \notin £ \wedge X^b_j \neq \varnothing)\}$

$CONF = \{ j \mid ((ta(s), j) \in £ \wedge X^b_j \neq \varnothing)\}$

In addition, given that $X^b_j = \{Z_{i,j}(\lambda^*(s_i)) \mid (i \in IMM \wedge i \in I_j)\} \cup \{Z_{self,j}(x) \mid x \in x^b \wedge self \in I_j\}$

$INF' = \{ j \mid ((i \in I_j \mid (ta(s),i) \in £) \vee (self \in I_j)) \wedge X^b_j \neq \varnothing)\}$

$INT' = \{ j \mid ((ta(s), j) \in £ \wedge X^b_j = \varnothing)\}$

$CONF' = \{ j \mid ((ta(s), j) \in £ \wedge X^b_j \neq \varnothing)\}$

$EXT' = \{ j \mid ((i \in I_j \mid (ta(s),i) \in £) \vee (self \in I_j)) \wedge (ta(s),j) \notin £ \wedge X^b_j \neq \varnothing)\}$

According to these new definitions, we can reformulate the resultant transition functions of the atomic cell space to include the events list handling. The only function that does not deal with these cell groups extracted from event list (£), is the external function. It just deals with the boundary cells that received external inputs through the cell space ports. At the end of every transition cycle of the atomic cell space, the model checks the events list (£) to see if it contains more scheduled events and if so, it extracts the list of cells with minimum time advance. On the expiration of that minimum time advance, the output function will access the *IMM* list and let the cells in this group send their output messages. Then, the internal or confluent transition functions are responsible to obtain the corresponding cell groups where the external transition function will work on the boundary cells that received external messages. Another source of speed up in the events list can be achieved by not letting the list to hold passive cells in its records. The following formulas shows how the transition functions are adjusted to handle the events list.

$\lambda(s)$: *for all* $i \in IMM$

Apply λ^* *to* $Cell_i$ [$Y_i = \lambda^*(s_i)$]

$\delta_{int}(s)$: *obtain cell groups INT, EXT, CONF*

Update £ : *time=time-ta(s) for all (time, i)* $\in$ £

delete any ev=(0 , i) where ev $\in$ £

for all $i \in INT$: *apply* δ^*_{int} *to* $Cell_i$

schedule (next ta, i)

for all $i \in EXT$: *apply* δ^*_{ext} *to* $Cell_i$

schedule (next ta, i)

for all $i \in CONF$: *apply* δ^*_{con} *to* $Cell_i$

schedule (next ta, i)

if (£ $\neq \{\}$)

extract IMM from £

ta(s) = minimum {time $\mid$ *(time, i)* $\in$ £ *}*

ACCEPTED MANUSCRIPT

else

clear IMM={}

ta(s) = ∞

$\delta_{con}(s,x^b)$: *obtain cell groups INT', EXT', CONF'*

Update £: *time=time-ta(s) for all (time, i)∈ £*

delete any ev=(0 , i) where ev∈ £

for all i∈ INT' : apply δ^*_{int} *to* $Cell_i$

schedule (next ta, i)

for all i∈ EXT' : apply δ^*_{ext} *to* $Cell_i$

schedule (next ta, i)

for all i∈ CONF' : *apply* δ^*_{con} *to* $Cell_i$

schedule (next ta, i)

if (£ ≠{})

extract IMM from £

ta(s) = minimum {time |(time, i) ∈ £ }

else

clear IMM={}

ta(s) = ∞

$\delta_{ext}(s,e,x^b)$: *Update £:* *time=time-e for all (time, i)∈ £*

for all i∈ { j | self∈$I_j \wedge X^b_j \neq \emptyset$*}*

apply δ^*_{ext} *to* $Cell_i$

schedule (next ta, i)

if (£ ≠{})

extract IMM from £

ta(s) = minimum {time |(time, i) ∈ £ }

else

clear IMM={}

ta(s) = ∞

4.4. Transforming Cells to Non-Modular Form

So far, all the above specifications are in the modular form, a major speed up can be achieved by transforming these cells into non-modular form. In non-modular form, a cell can access (read) the state variables of its neighbors and there is no need for message passing through ports. In contrast to multi-component DEVS [6], the implementation we are seeking here allows cells to read each other's states, but they are only allowed to make their

ACCEPTED MANUSCRIPT

own state transitions. In case a cell changes its state, its neighboring cells need to be added to the cell group *EXT* instead of allowing the cell itself to change their states directly.

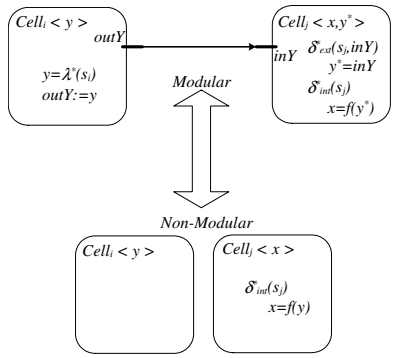

Figure-2: Switching between modular and non-modular forms.

Figure-2 shows how models can be transformed from modular into non-modular form and vice versa. Transforming to non-modular form can be achieved by removing the ports from the atomic models and letting them directly access the neighboring models to read their state variables. In cellular space models, this will reduce the structure of the cell units into a smaller one that just stores states and variables only and it has no functions or processing done at its level. However, we need to keep the coupling relations so that each cell knows which neighboring cells to access. In this case, we can add to the cell structure a list of the neighboring cells (n) which can be accessed by that specific cell. In cellular space models, this list is the set of influencers which is equivalent to the influencees set.

$Cell_i = \langle X_i, S_i, Y_i \rangle \rightarrow Cell_i = \langle S_i, n_i \rangle$, *Where* $n_i = \{ j \mid j \in I_i \} = \{ j \mid i \in I_j \}$

Since we removed the ports from the cells, the cell space atomic model functions need to be redefined according to the new changes. The first change is that we do not need cells to generate outputs to ports since their neighboring cells can access their state variables directly. Therefore, when a cell goes through a state transition, its neighboring cells are added to the set of cells, *EXT*, that should fire their external transition functions.

Assumption-1:

The output values that are sent via messages by a cell in the modular form are actually values of one or more of its state variables.

Applying this assumption requires that we first define the state variables of each cell as follows: Given that each cell has a state set S_i, and each state $s_i \in S_i$ is a collection of values of the state variables that represent the current state of the cell i. Therefore, $s_i = (sv^1_i, sv^2_i, sv^3_i, \ldots sv^n_i)$ where n is the number of state variables in the cell. Note that the primary states (e.g. passive, active … etc) are also treated as one of the state variables which contain the name of the state as a string.

ACCEPTED MANUSCRIPT

The resultant non-modular cell space can be shown to be equivalent to the modular counterpart as follows: Given that assumption-1 is satisfied, each modular cell *i* will send its own values of the state variables through Y^b_i to its neighbors whenever there is a change in those values. Then, that neighboring cell *j* receives those values at X^b_j. Therefore, $Y^b_{i,j} = X^b_{i,j}$ for all cells $i,j \in D$ and there exist coupling relation $Z_{i,j}$. However, the initiating cell, *i,* actually sends its own selected set (v_{ij}) of state values to neighbor *j* and so, $Y^b_{i,j} = \{sv^k_i \mid (k \in v_{ij}) \wedge (1 \le k \le n)\}$ which represent the set of state values that should be sent to cell *j*.

Then, given that there exist the coupling $Z_{i,j}$ where $i,j \neq self$ *(i.e. not a boundary coupling)*,

$$X^b_{i,j} = Y^b_{i,j} = \{sv^k_i \mid (k \in v_{ij}) \wedge (1 \le k \le n)\},$$

$$Y^b_i = \underset{i \in I_j}{\times} Y^b_{i,j} = \underset{i \in I_j}{\times} \{sv^k_i \mid (k \in v_{ij}) \wedge (1 \le k \le n)\} = \{sv^{k11}_i, sv^{k12}_i, sv^{k13}_i, ..., sv^{k21}_i, sv^{k22}_i, sv^{k23}_i, ...\}$$

where for each receiving cell (*j*) there are multiple state variables to be received denoted here as (*kjw*) where $kj1, kj2, kj3, ... \in v_{ij}$, for example, k21 means the index (in sv_i) for the first value to be sent to cell number 2.

$$X^b_j = \underset{i \in I_j}{\times} X^b_{i,j} = \underset{i \in I_j}{\times} \{sv^k_i \mid (k \in v_{ij}) \wedge (1 \le k \le n)\} = \{sv^{k11}_i, sv^{k12}_i, sv^{k13}_i, ..., sv^{k21}_i, sv^{k22}_i, sv^{k23}_i, ...\}.$$

That means that all input/output values are equivalent to the state variables of the cells. Therefore, we can redesign the cell space model to make it fully non-modular by making each cell access the required state variables of its neighboring cells while still keeping the equivalency given that assumption-1 is met. According to Figure-2, implementing modular cells makes each cell keep records of its neighboring state variable (y^*) where, in the non-modular from, there is no need to keep a record since every time the cell needs a value from its neighbor (y), it access it directly. This will make the internal cell transition function defined for state variables of each cell as well as the state variables of its neighboring cell since it has access to all of them according to the coupling relation.

For boundary cells:

$$X^b = \underset{self \in I_i}{\times} X^b_{self,i} = \underset{self \in I_i}{\times} \{sv^k_x \mid (k \in v_{xi}) \wedge (1 \le k \le n)\}$$

$$Y^b = \underset{i \in I_{self}}{\times} Y^b_{i,self} = \underset{i \in I_{self}}{\times} \{sv^k_i \mid (k \in v_{iy}) \wedge (1 \le k \le n)\}$$

Where v_{xi} is the set of state variables needed to be received by boundary cells through external input ports and v_{iy} is the set of state variables needed to be sent by boundary cells through external output ports. The state variables sv^k_x are used as storage for the external values that are received by the cell space through input ports and they are only accessed by the boundary cells. On the other hand, the external outputs that are required to be sent out of the cell space are the states variables of the boundary cells.

One more issue in the equivalency to the non-modular form is that the neighboring cells do not always have the last updated values. One reason for that is using the quantized DEVS in which the cell does not inform the neighboring cells with its last modification if the difference is not above a specific quantum. For example, in differential equation models, there is a rate of change in some state variables which means the variable is

ACCEPTED MANUSCRIPT

continuously changing but, in quantized DEVS, the change take place only when there is time advance expiration or an external event occurred. The above equivalency analysis is correct if we set the quantum to zero. However, if it is not zero, each cell should keep two copies of state variables (e.g. now and new). Whenever a cell needs to access a value in its neighboring cell, it will access the (now) value which represents the last value that crossed the quantum level (i.e. was sent to the cell through ports in the modular form). The (new) value is the last updated value of the cell which is kept different from (now) till it crossed the quantum level and the change will be committed to (now) to be available for other cells to access. Now, we can redefine the cell transition functions as follows:

For each specific cell i :

δ_{int}: $S_{i\text{-}now} \times S_{i\text{-}new}$ *where for both old and new* $s_i \in S_i$ *given* $s_i=(sv^1_i, sv^2_i, sv^3_i, \ldots sv^n_i)$

$$\delta_{ext}: \bigtimes_{j\in I_i} S_{j-now} \times Q_i \rightarrow Q_i$$

$$\delta_{con}: \bigtimes_{j\in I_i} S_{j-now} \times Q_i \rightarrow Q_i$$

λ: $S_{i\text{-}new} \rightarrow S_{i\text{-}now}$

The output function for each cell just updates the state variables of the current cell in case it exceeds the quantum level and there is no need to generate any outputs if the cell is not a boundary cell. This task is actually done in the internal transition function δ_{int} and we can select not to duplicate the task. Another reason is to keep the new specification consistent with the DEVS specification where the output function is used to send messages only and does not initiate state or variable changes in the model. Therefore, δ_{int} will commit the changes in variables and add the neighboring cells to the scan group *EXT*.

4.5. Final Non-Modular Composed Format

The last detailed specification above shows that the cell internal output function tasks were encapsulated under the cell space output function and there is no need to define λ^* as an independent function. Similarly, δ_{ext} or δ_{con} are not defined for non-modular cells as shown in Figure-2.

Assumption-2:

The modular cells use δ_{ext} and δ_{con} to update the values of their neighboring states and then apply δ_{int} to make calculations and transitions according to the updated values. This means that δ_{ext} and δ_{con} are designed not to make calculations or processing, but force the cell to do an internal transition which considers the new updates.

$$\delta_{ext}(s_i,e_i,x^b_i)=(\{\text{“re-calculate”}, sv_i^1, sv_i^2, \ldots, sv_i^n\}, 0, \bigtimes_{j\in I_i} \{sv_j^k \mid (k \in v_{ji}) \wedge (1 \le k \le n)\} = x^b_i)$$

ta(“re-calculate”)=0;

$$\delta_{con}(s_i,x^b_i) = \delta_{int}(\delta_{ext}(s_i,ta(s_i),x^b_i))$$

ACCEPTED MANUSCRIPT

In the non-modular form, each cell will update its own values using the internal transition function δ_{int} and then, their neighboring cells will be added to the scanning list which in turn schedules an external event for the neighboring cells. This is exactly what the cell output function λ^* and the cell external transition function δ_{ext} do given that assumption-2 is satisfied. Therefore, the equivalency to the modular form is satisfied and there is no need to have the functions λ^*, δ_{ext}, and δ_{con} explicitly in the resultant model since their tasks are already implied in the new framework. As a result, the model can be simplified for ease of user specification in such a way that each cell has only an internal transition function that is applied every time a cell becomes active. In addition, the cell groups can be merged as we do not distinguish between the different groups. However, in addition to the *IMM* group, we still need the *EXT* group in order to update the neighboring values of the imminent cells. Note that from now on, we will refer to the cell internal transition function δ_{int} as the cell's local transition function Δ^* ($\Delta^*=\delta_{int}$).

For each specific cell i :

$\Delta^*: \underset{j\in I_i}{\times} S_{j-now} \times S_{i\text{-}now} \rightarrow S_{i\text{-}new}$ *where* $s_i \in S_i$ *given* $s_i=(sv^1_i, sv^2_i, sv^3_i, \ldots sv^n_i)$

An atomic cell space now consists of identical fully non-modular cells having the same transition function at the cell space level but covering different sets of data and state variables. Finally, the cell space was converted into an atomic non-modular P-DEVS given that assumption-1 and assumption-2 are satisfied. The resultant model can be simplified and put in the following format:

$\lambda(s)$: *for all* $\{ i \mid i \in IMM \wedge i \in I_{self} \}$

$Y = Y \cup Z_{i,self}(\{ sv_i^k \mid (k \in v_{iy}) \wedge (1 \le k \le n) \})$

$\delta_{int}(s)$: *Update* £: $time=time-ta(s)$ *for all* $(time, i) \in$ £

delete any $ev=(0 , i)$ *where* $ev \in$ £

for all $i \in IMM$

if $s_{i\text{-}new} - s_{i\text{-}now} > quantum$

for all $\{ i \mid i \in I_j \wedge j \neq self \}$ $EXT=EXT \cup \{ j \}$

$s_{i\text{-}now} = s_{i\text{-}new}$

for all $i \in \{IMM \cup EXT\}$:

apply Δ^* *to* $Cell_i$

schedule (next ta, i)

clear cell group EXT={}

if (£ $\neq \{\}$)

extract IMM from £

$ta(s) = minimum \{time \mid (time, i) \in £ \}$

else

clear IMM={}

$ta(s) = \infty$

ACCEPTED MANUSCRIPT

$\delta_{con}(s,x^b)$: $conf = true$

$\delta_{ext}(s,e,x^b)$

$\delta_{int}(s)$

$conf = false$

$\delta_{ext}(s,e,x^b)$: if (!conf) Update £ : $time=time-e$ for all (time, i)∈ £

$EXT = \{ j \mid self \in I_j \wedge Z_{self,i}(X^b \neq \emptyset)\}$

for all $i \in EXT$

$S_{self\text{-}now} = S_{self\text{-}now} \cup Z_{self,i}(X^b)$

if (!conf) $IMM = EXT$

$ta(s)=0$ // fire $\delta_{int}(s)$ to make the state transitions

4.6. A Proposition to Show the Generality of the Approach

The generality of the approach that follows the closure under coupling property in parallel DEVS was ensured in all steps in the procedure with no constraints except the two assumptions introduced above. The final format reached assumes that the models satisfy those assumptions. This section shows and proves the generality of the approach that spans all cellular DEVS models: even the ones that do not agree with the assumptions. We show that any modular cell in cellular DEVS model can be modified to satisfy the two assumptions as follows.

A modular cell (i), that does not satisfy assumption-1, sends an output value y that is not among the cell's state variable values s_i where $s_i \in S_i$. That value will either be a fixed value for each cell, fixed parameter, or cell's calculated value where $y \notin s_i$. It was shown above that $s_i=(sv^1_i, sv^2_i, sv^3_i, \ldots sv^n_i)$ where n is the number of state variables in the cell model. To satisfy assumption-1, the modular cell (i) can be updated to extend the state variables representation in order to include one more state variable that account for the values y. As a result $s_i=(sv^1_i, sv^2_i, \ldots sv^{n+1}_i)$ where sv^{n+1}_i is a newly added state variable to the model that stores the value of y which can then be sent through external ports according to assumption-1.

A modular cell (i), that does not satisfy assumption-2, does some computations and state transitions in δ_{ext}. Operations in this function can be separated into two phases in any DEVS cell model. The first one accepts the external values on ports and updates the state variables accordingly in zero time. The other phase will include the calculations and state transitions that are based on the updated values.

$\delta_{ext}(s_i,e_i,x^b_i) = s_{i\text{-}new} \rightarrow \delta_{ext}(s_i,e_i,x^b_i) = \delta_{ext}(s_{i\text{-}temp},e_i) = s_{i\text{-}new}$

First phase:

$(\underset{j \in I_i}{\times} \{sv^k_j \mid (k \in v_{ji}) \wedge (1 \leq k \leq n)\} = x^b_i) + s_i \rightarrow s_{i\text{-}temp}$

$s_{i\text{-}temp}$ = {"re-caculate", $sv_{i\text{-}temp}{}^1, \ldots, sv_{i\text{-}temp}{}^n$ }

Second phase:

$\delta_{ext}(s_{i\text{-}temp},e_i) = s_{i\text{-}new} = \{ sv_{i\text{-}new}{}^0, sv_{i\text{-}new}{}^1, \ldots, sv_{i\text{-}new}{}^n \}$.

ACCEPTED MANUSCRIPT

To satisfy assumption-2, the second phase should be moved into the internal transition function that should now deal with a temporary state "re-calculate". The first phase is left in the external function to update state variables based on the received messages. Therefore, the external transition function can be defined in the following format:

$\delta_{ext}(s_i, e_i, x^b_i) = (\{\text{"re-calculate"}, sv_i^1, sv_i^2, \ldots, sv_i^n\}, 0, \underset{j \in I_i}{\times} \{sv_j^k \mid (k \in v_{ji}) \wedge (1 \le k \le n)\} = x^b_i)$

which account for the new updates and sets the time advance to be zero in order to fire the internal transition function in the next step that will deal with the temporary state "re-calculate" on the updated values and hence satisfying assumption-2. Similar approach can be done in the confluent function.

5. Fast Cellular DEVS Specification

Based on the full composition process, we can now introduce a specification (i.e. formalism layer) to represent atomic DEVS cell space that will run faster than the conventional implementations that are based on representing each cell as an atomic model. Therefore, a cell space can be formed in the following P-DEVS atomic structure:

Atomic $CellSpace = \langle X, S, Y, D, B, \{Cell_{id}\}, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, £ \rangle$.

Where,

D is the set of cell ID's encapsulated in this atomic model, |D| = number of cells in the model

B is the set of the boundary cells ID's

$Cell_{id} = \langle n, S^* \rangle$ for all $id \in D$ and n is the set of neighboring cells and S^* the state variables set (in non-modular form) where $s^* = (sv_1, sv_2, sv_3, \ldots..) \mid sv_n$ is the value of the n^{th} state variable of the cell for a given $s \in S^*$.

X is a set of input values defined only for boundary Cells $id \in B$ (set of ports/values in coupled structures).

Y is a set of output values.

S is a set of general states of the atomic model

The total state set $Q = \{(s, e, \{\underset{id \in D}{\times} Q^*_{id}\}) \mid s \in S, 0 \le e \le ta(s)\}$

$Q^*_{id} = \{(s^*_{id}, e_{id}) \mid s^* \in S^*, 0 \le e_{id} \le ta(s^*)_{id}\}$

δ_{int}: $Q \to Q$ is the *internal transition* function.

δ_{ext}: $Q \times X^b_B \to Q$ is the *external transition* function,

δ_{con}: $Q \times X^b_B \to Q$ is the *confluent transition* function,

λ: $Q \to Y^b$ is the output function, only for cells with $id \in B$

£: is the next events list

$ta = min\{time \mid (time, id) \in £\}$

The four functions are executed iteratively and efficiently as explained at the end of section 3.3.3.

For multi cell spaces, we can couple more than one atomic cellular space in one coupled model which will be in the form of parallel coupled DEVS:

ACCEPTED MANUSCRIPT

$CM = \langle X, Y, D, \{CellSpace_i\}, \{I_i\}, \{Z_{i,j}\} \rangle$.

In addition, the specification can be extended to d-dimensional cell space as follows:

$CellSpace^d = \langle X, S, Y, \{N_i\}, d, D, B, \{Cell_{id}\}, \delta_{\text{int}}, \delta_{ext}, \delta_{con}, \lambda, £ \rangle$.

Where, d is the dimension of the cell space and N_i is the number of cells in the ith dimension given $1 \leq i \leq d$. Then, id will be a set of IDs for cell in each dimension $\{id_i\}$. In array implementation of the cell space atomic mode, this update will result in d-dimensional state arrays for the state variables. However, in that case, the cell's id will be the set of array indexes in each dimension.

6. SLOPE CRITICALITY MODEL

Slope criticality has many applications in dynamical natural phenomena like sand piles, debris flows, landslides, and land surface evolutions (e.g. [26, 27]). In such models a local piece of spatial space is stable in certain slopes till it exceeds a critical slope when it starts initiating a mass flow to the neighbors. This flow might continue propagating over neighboring areas forming a bigger global flow that keeps running until it reaches global stability.

Applying cellular space modeling techniques to these models will result in dividing the spatial space into cells in which each cell will apply automata rules to decide its own local state. This will be based on the slope calculation which is represented , as shown in Figure-3 by the height difference (ΔZ) between each two neighboring cells given that the distance between them is fixed (d_{ij}). When a cell has a slope greater than a certain critical value, it initiates a mass flow to its neighbor with a fixed rate R.

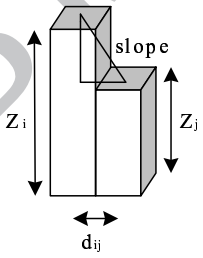

Figure-3: Slope Calculation.

In this two dimensional cellular model, we apply the 4 neighbors rule. Criticality check involves calculating the slopes between a cell and its four neighbors. Whenever a cell finds that the slope at any direction exceeds the known threshold, it calculates the flow rate for that direction and informs the neighbors with its outgoing flow. The neighboring cells then update their states according to the incoming flows. Accordingly, the height of the initiating cell will be reduced by the amount of increase in the receiving neighbor and this conveys the mass conservation. The simulator will iterate until there is no more cells in critical state or by reaching a predefined stopping rule.

7. MODEL EQUIVALENCY TEST

The new approach presented in this work is based on generating an equivalent atomic model of the conventional coupled cell space approaches. Before running experiments and making conclusions, the new generated atomic models must be made identical to the original coupled DEVS implementation. In this work, the simulation verification method is followed in order to dynamically check the equivalency of the two approaches. Figure-4

ACCEPTED MANUSCRIPT

illustrates the idea of dynamically comparing simulation runs of two systems. A DEVS atomic model is employed to work as a comparator that monitors the state trajectories generated by both systems and generates a fail report if they are not equivalent. The report should include the non-equivalent values generated as well as their time. If no fail report is generated and the verification run comes to an end, the two systems can be declared identical. In case of non-terminating simulations, the larger number of iterations employed, more is the confidence that will be obtained in the conclusions.

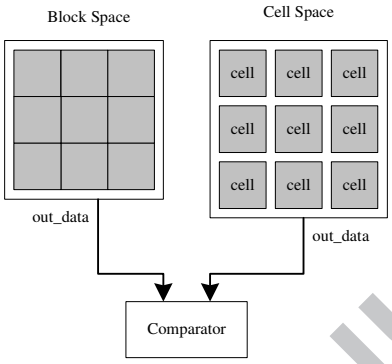

Figure-4: Simulation approach of equivalency verification.

8. EXPERIMENTAL RESULTS

The slope criticality model explained above was implemented in DEVSJAVA using the new approach for the purpose of performance comparison with the conventional cellular DEVS implementations. A 32 by 32 cellular space model was run with different setups. In all cases, identical data (i.e. heights and flows at any given time) were generated which ensures the equivalence of models in the new approach as well as the conventional approach.

By implementing the new method in DEVSJAVA, we achieved an enormous time reduction, messages reduction, as well as number of simulator iterations reduction as shown in Table 1. These new findings are not in favor of the conventional way of implementing cellular automata in DEVS especially when running on a single machine. For example, implementing the new approach by encapsulating all cells in one atomic model gives a speed up of 46 in execution time over the conventional approach by saving 121569 message-handling simulator iterations using the conventional DEVSJAVA simulator (DJSim).

Table 1. Experimental Results.

	Message Reduction (%)	Iterations	Execution Time (Min)	
			DJSim	ADJSim
Conventional Cell DEVS	0	171478	63.17	16.56
New Cell DEVS	100	49909	1.36	1.36

In Table 1, the last column shows the execution time when repeating the runs over the implemented activity-based DEVSJAVA simulator (ADJSim) suggested in [2]. This simulator alone achieved a speed up of 3.8 over the conventional simulator and noticeable speedups when combined with our approach. The last case shows that the

ACCEPTED MANUSCRIPT

new simulator has no gain because the model consists of single cell which is always active during the simulation run.

9. CONCLUSION

Conventional modular approaches of modeling cellular DEVS models were found to poorly perform in the case of very large scale spaces with high cell activities. Some related works were done to speedup simulations by the means of simulator enhancements that deal efficiently with the big volume of communication messages. Despite all of these enhancements, the models still spend a large amount of computational time in dealing with messages rather than spending all computational efforts in the actual model tasks.

In this paper, a new specification was introduced to specify the cellular DEVS models in an efficient non-modular form. The new formalism was formulated using the closure under coupling property of DEVS in order to ensure equivalency of the models to their modular counterparts in parallel DEVS. In this new methodology, a cell space can be composed entirely into one or more atomic models where each contain a set of non-modular cells and run as a standard parallel DEVS model in term of performing external and confluent operation in addition to the internal transitions. Non-modular Models that were developed using the new cellular DEVS specification were found to outperform their modular equivalents. The speedup was gained from two sources. The first one is the efficient scanning of active cells which also can be achieved using simulator enhancements. The other one is the elimination of the inter-cell messages by fully composing the cellular space model into atomic one.

Specifying large and complex cellular models using the new specification was found to be complicated and difficult to verify. Therefore, further research work is already under progress to automate and ease the process of model development and verification. One proposed solution is to develop a multi-layer cell DEVS representation in which the composition task is done automatically in a hidden layer away from the model developer. In addition, parallel and distributed simulation is another aspect of performance investigation at implementation level since the parallel specification is ensured through the equivalency to parallel DEVS models. As a result, models developed with the new specification can be run in any parallel and/or distributed DEVS environment.

REFERENCES

[1] G. Wainer, "Modeling and simulation of complex systems with Cell-DEVS," in *WSC '04: Proceedings of the 36th conference on Winter simulation* Washington, D.C.: Winter Simulation Conference, 2004, pp. 49-60.

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

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

[5] 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.

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

ACCEPTED MANUSCRIPT

[7] W. B. Lee and T. G. Kim, "Simulation Speedup for DEVS Models By Composition-based Compilation," in *Summer Computer Simulation 2003*, 2003, pp. 395 - 400.
[8] T. Beltrame, "Design and Development of a Dymola/Modelica Library for Discrete Event-oriented Systems Using DEVS Methodology," in *Department of Computational Science*. vol. Master of Science: ETH Zurich, 2006.
[9] F. A. Shiginah and B. P. Zeigler, "Transforming DEVS to non-modular form for faster cellular space simulation," in *2006 DEVS Symposium*, 2006, pp. 86-91.
[10] A. C.-H. Chow, "Parallel DEVS: a parallel, hierarchical, modular modeling formalism and its distributed simulator," *Trans. Soc. Comput. Simul. Int.,* vol. 13, pp. 55-67, 1996.
[11] B. Chopard and M. Droz, *Cellular automata modeling of physical systems*: Cambridge University Press, 1998.
[12] T. Toffoli and N. Margolus, *Cellular automata machines: a new environment for modeling*. Cambridge, MA, USA: MIT Press, 1987.
[13] T. N. Mudge, R. A. Rutenbar, R. M. Lougheed, and D. E. Atkins, "Cellular image processing techniques for VLSI circuit layout validation and routing," in *DAC '82: Proceedings of the 19th Design Automation Conference* Piscataway, NJ, USA: IEEE Press, 1982, pp. 537-543.
[14] D. Talia, "Cellular Processing Tools for High-Performance Simulation," *Computer,* vol. 33, pp. 44-52, 2000.
[15] B. D. Malamud and D. L. Turcotte, "Cellular-Automata Models Applied to Natural Hazards," *Computing in Science and Engineering,* vol. 2, pp. 42-51, 2000.
[16] X.-S. Yang, "Characterization of multispecies living ecosystems with cellular automata," in *ICAL 2003: Proceedings of the eighth international conference on Artificial life* Cambridge, MA, USA: MIT Press, 2003, pp. 138-141.
[17] R. Hu and X. Ruan, "Differential equation and cellular automata model," in *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, 2003, pp. 1047-1051.
[18] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.
[19] N. Ganguly, P. Maji, S. Dhar, B. K. Sikdar, and P. P. Chaudhuri, "Evolving Cellular Automata as Pattern Classifier," in *ACRI '01: Proceedings of the 5th International Conference on Cellular Automata for Research and Industry* London, UK: Springer-Verlag, 2002, pp. 56-68.
[20] T. Yu and S. Lee, "Evolving Cellular Automata to Model Fluid Flow in Porous Media," in *EH '02: Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)* Washington, DC, USA: IEEE Computer Society, 2002, p. 210.
[21] M. V. Avolio, G. M. Crisci, D. D'Ambrosio, S. Di-Gregorio, G. Iovine, R. Rongo, and W. Spataro, "An Extended Notion of Cellular Automata for Surface Flows modeling," *WSEAS Transactions on Computers,* vol. 2, p. 6, 2003.
[22] S. Wolfram, *A New Kind of Science*: Wolfram Media, 2002.
[23] R. O. Cunha, A. P. Silva, A. n. A. F. Loureiro, and L. B. Ruiz, "Simulating Large Wireless Sensor Networks Using Cellular Automata," *Simulation Symposium, Annual,* vol. 0, pp. 323-330, 2005.
[24] G. wainer, "Performance Analysis of Continuous Cell-DEVS Models," in *Proceedings of High Performance Computing & Simulation (HPC&S) Conference; 18th European Simulation Multiconference*, Magdeburg, Germany, 2004.
[25] H. Saadawi and G. Wainer, "Modeling a sand pile application using Cell-DEVS," in *Proceedings of the 2003 Summer Computer Simulation Conference*, Montreal, QC. Canada, 2003.
[26] P. Bak, *How Nature Works: The Science of Self-Organized Criticality*: Springer-Verlag Telos, 1996.
[27] C. Moore and M. Nilsson, "The Computational Complexity of Sandpiles," *Statistical Physics,* vol. 96, pp. 205-224, 1999.