

The Activity tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems

Alexandre Muzy¹, Rajanikanth Jammalamadaka², Bernard P. Zeigler², James J. Nutaro³

¹LISA CNRS
Università di Corsica – Pasquale Paoli
Campus Grossetti, 20250 Corti, France
Email: a.muzy@univ-corse.fr

²Arizona Center for Integrative Modeling and Simulation
University of Arizona
Tucson, AZ, United-States of America
Email: {rajani,Zeigler}@ece.arizona.edu

³Oak Ridge National Laboratory
Oak Ridge, TN, United-States of America
Email: nutarojj@ornl.gov

Abstract

From a modelling and simulation perspective, studying dynamic systems consists of focusing on changes in states. According to the precision of state changes, generic algorithms can be developed to track the activity of sub-systems. This paper aims at describing and applying this more natural and intuitive way to describe and implement dynamic systems. Activity is mathematically defined. A generic application case of diffusion is experimented to compare the efficiency of quantized state methods using this new approach with traditional methods which do not focus computations on active areas. Our goal is to demonstrate that the concept of activity can estimate the computational effort required by a quantized state method. Specifically, when properly designed, a discrete event simulator for such a method achieves a reduction in the number of state transitions that more than compensates for the overhead it imposes.

Introduction

Recently, a new research direction has been discussed in [1] for component-based simulation and applied to complex natural systems [2]. This approach aims at defining a new framework for tracking activity of components in dynamic simulations. Discrete-time and discrete-event implementations of this technique can be found in [3-6]. In this paper, our aim is to apply the activity tracking paradigm to an example in which analytic solutions can be obtained to enable comparison of activity-tracking solution techniques with conventional ones along the dimensions of both execution efficiency and accuracy. A one-dimensional partial differential equation system is discretized using the quantization method presented in [7]. The efficiency, stability, and accuracy of the method examined in this paper, and of several related methods, have been studied in [8-13]; these papers discuss extensions of the quantization concept to stiff systems and to higher order numerical integration methods. However, [8] did not explicitly consider the relationship of the quantization technique to activity

tracking. Our aim here is to show how the activity of spatial systems can be tracked using discrete-events and to explain the results of [8] in these terms. Furthermore, a new formal concept is introduced to discuss discrete-event implementations of continuous systems: the activity measure. This activity measure [7, 14] supports predicting the number of computations (or boundary crossings) necessitated by quantization to approximate a continuous curve.

In the rest of this paper, first the quantization and activity tracking paradigm concepts are introduced. Then, these concepts are applied to a one-dimensional diffusion process. Next, simulation results are discussed and presented. Finally, conclusions and perspectives are given.

1 Background

Activity of systems can be tracked from both modeling and simulation perspectives. From the simulation perspective, messages and computations should be tracked in the whole distributed system without introducing errors. From the modeling perspectives, the detection precision of state changes can be adapted to reduce message exchanges and computation introducing a controlled error.

1.1 Simulation perspective

There are three common types of discrete event simulation strategies, also called world views, that are employed in discrete event simulation languages and packages [15-17]: event-scheduling, activity-scanning, and process-oriented. A strategy makes certain forms of model description more naturally expressible than others. In all of these world-views, an *event* is an instantaneous change in the state of a system at a particular time. Event scheduling models work with pre-scheduling of all events and there is no provision for activating events by tests on the global state. In contrast, in the activity scanning approach, events can be conditioned on a contingency test in addition to being scheduled to occur in time. A model is said to be *active* when both its scheduling time has occurred and its contingency test is satisfied. The *process interaction* world view is a combination of the event scheduling and activity scanning strategies. A detailed formulation is provided in [16].

In simulations, a distinction is usually made between the continuous time of reality and simulation time. Simulation time can be managed two ways [15]: by a clock or by discrete-events. In a clock (or discrete-time) driven simulation, time is incremented by a step Δt , from t to $t+\Delta t$. State changes occurring between $[t, t+\Delta t]$ are computed at $t+\Delta t$. The time base is called discrete (or synchronous) and the models are called discrete-time models. In a discrete-event driven simulation, the simulation progresses from the occurrence time of one event to the occurrence time of the next event, *i.e.*, from one state change to the next. The time base is called continuous (or asynchronous) and the models are called discrete-event models.

The advantage of discrete-event driven simulations is that a simulation model evolves directly from one state change to another. No computations are performed during inactive periods, but this requires that the next state change can be forecast from the current state and it requires an efficient method for scheduling events. On the other hand, in a discrete-time driven simulation one has to cope with the limited precision of the time step and the consequent difficulties with detecting and acting on asynchronous events.

The appropriate choice of a time management scheme depends on the nature of the system and on the modeling objectives. For a system in which every state change occurs at a Δt , discrete-events will produce

simulation overhead, and a discrete-time driven simulation will be more efficient because we do not have to predict (by computation) what we already know will happen and when. However, we can easily admit that in natural systems discrete-time evolution does not exist. Discrete-time flows only exist in a modeler's head and in industrial processes (*e.g.*, robots) or after the discretization of real continuous time by humans [18]. For other systems, "while other formalisms allow representation of space and resources, only discrete-event models offer the traditional ability to explicitly and flexibly express time and its essential constraints on complex adaptive systems behavior and structure" [19].

The activity-tracking paradigm emerges naturally from a discrete-event approach to simulation. Discrete-events, however, are essential to take into account external events in discrete-time components. Moreover, every discrete-time increment can be considered as an internal event of a simulation model. With this perspective it is possible to usefully apply activity-tracking techniques to discrete-time simulations.

Figure 1 depicts the essential parts of an activity-tracking simulation. It merges the three usual world-views (activity-, event-, and process-oriented strategies) into a single framework that includes discrete-time driven models as well; the usual world-views are underlined where they appear in the activity-tracking strategy. Marks are added to, and removed from, components to track activity as it propagates through the model; these marks are used to maintain the set of active components as the simulation progresses. Every activity-tracking simulation has three basic steps:

- **Step I:** Components exchange information and the propagation of activity is tracked. The current active set is scanned for components that have output events and these events are routed to their destinations; routing in hierarchal models can be done recursively [4]. The active set is updated to include atomic components that have received these output events. Atomic components are basic behavioral components that are coupled together to build a hierarchical structure [16].
- **Step II:** New states are computed for active components from their current states and inputs. Components that changed state significantly are marked and added to the active set; components that do not undergo a significant change of state are removed from the active set. In a discrete-event driven simulation, the active set is an event scheduler and the active components are in the schedule or will receive an input from a model in the schedule.

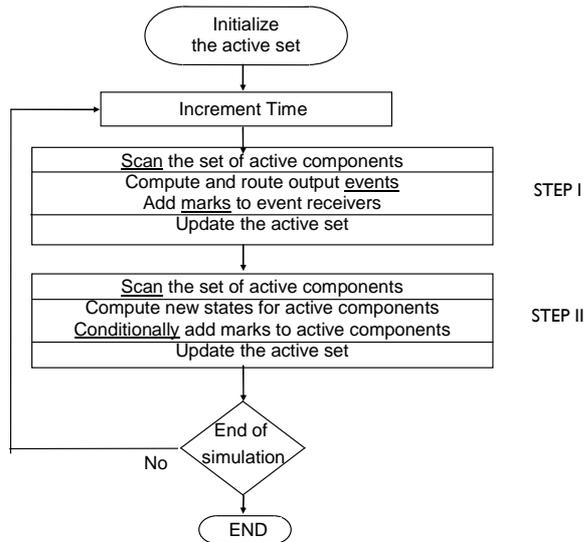


Figure 1. State-centric activity tracking pattern

This is a state-centric two-step pattern. Both local and global levels of modularity for state transitions can be achieved. Local transitions are achieved on states of single components. Global transitions are achieved on many component states. These specification levels are discussed in the next sub-section.

1.2 Modeling perspective

Significant success has been achieved in applying Discrete Event Simulation approaches to simulate continuous systems that are characterized by spatiotemporal heterogeneity in their activities [6, 7, 20]. A Discrete Event approach makes effective use of the computational resources by allocating them to regions in proportion to their activity.

For simulating the diffusion process, specific algorithms have been proposed to track activity (changes in the value of cells) in space [21] based on discrete-time discretization schemes. However, these methods are based on a discrete-time numerical method using a specific activity tracking algorithm work by dynamically adjusting the calculation domain. When the algorithm is applied to another problem, it is impossible to predict its simulation efficiency and, consequently, whether or not the algorithm is applicable to the new problem.

Quantization-based numerical methods work, not by adjusting the computational domain, but by allowing the advance of time to be determined locally at each grid point. Quantization is a recent numerical method [16, 22]. This method is used to track activity in time and space [5, 7] thus reducing the number of computations necessary to approximate a continuous curve. In contrast to discrete-time numerical methods, that discretize time in equal time steps, quantization works by discretizing (or quantizing) the state variables of a continuous system in equal quanta. The quantized model is then implemented in DEVS. Roughly speaking, in such a simulation, discrete events occur at quantum boundary crossings; the time advance (i.e., time to go from one event to the next) at any cell is inversely proportional to the current temporal derivative at that cell.

1.2.1 Quantization methodology

Quantization is part of the DEVS framework. The latter is constituted of a set of abstract algorithms, mathematical structures, notations... to be used for modeling and simulating components of systems. Discretization methods are usually used as a technique to obtain algorithms for discretizing continuous systems. These algorithms are then encoded from scratch. DEVS constitutes a framework for this implementation. We consider here this framework through the quantization technique. The latter offers a new way of considering continuous systems. However, a clear distinction needs to be made with other adaptive discretization methods.

No confusion should arise between quantization based methods and Adaptive mesh refinement techniques. The Adaptive mesh refinement techniques use a global time step. The time step can be adaptive in the sense that the time step changes from iteration to iteration, but the time step is applied uniformly across space. The spatial and temporal grids are discretized simultaneously and in the same way, and then the variation of the state variables is tracked in space and time. Discrete event techniques differ from adaptive mesh refinement because the time advance is not applied uniformly across space. Rather, it is entirely local, with different spatial grid cells having different rates of advance. In effect, each grid point has its own adaptive time step that can be different from the steps used by other grid points. Many adaptive mesh refinement methods [23, 24] are based on focusing work in a manner related to activity, but activity is left as an informal concept.

Recently, interest has developed in locally adapting simulation time steps for new classes of models arising in bioinformatics and other areas. Although such methods are similar to the approach discussed here, they differ in certain key respects. Logg [25] embeds the finite element method and Galerkin methods into a mathematical framework that supports “multi-adaptive time” integration. The method requires a priori and a posteriori estimates of the simulation errors. [26] proposes meta-algorithms based on discrete-event schedulers and the Hermite polynomial interpolation for “multi-algorithm, multi-timescale methods” for cell simulation. Mathematical structures of meta-algorithms are provided for management of sub-algorithms in a non-modular way. In contrast, the framework discussed here constitutes a computationally-based method founded on DEVS. The advantage of this framework resides in its: (i) application simplicity and (ii) simulation efficiency. Specifically, the method does not require specific knowledge of the model to adapt the local time advance in a spatially explicit model, and simulation efficiency derives directly from the focus on events rather than time step scanning methods.

1.2.2 Activity measure of continuous systems as an idealization of a quantized integrator

The activity principle has been defined in [2, 27]. Activity, as it relates to quantization, is defined as the rate of change of the parameter in the temporal and spatial dimensions. The following is the definition of activity for a continuous segment. It provides a precise measure of the computational effort required by an ideal quantized method. In fact, it is a reasonable estimate of the computational effort required by implementations of these methods as well, and so provides a practical basis for anticipating where these methods might be useful.

In Figure 2, D corresponds to the quantum (the minimum threshold for change below which no processing occurs) and the m_i corresponds to the maxima and minima of the curve, where the first and last m_i are the values of the function at the initial and final times.

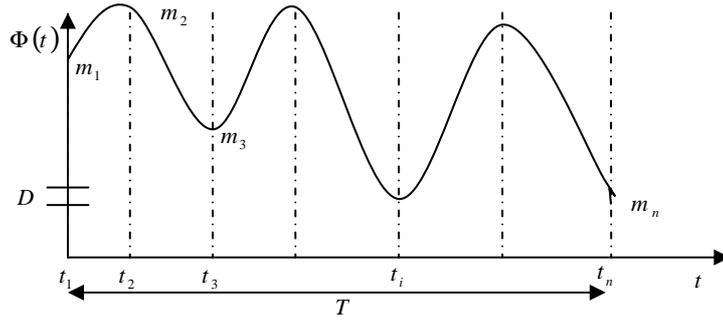


Figure 2. Definition of Activity

The *activity* in an interval $[0, T]$ is defined as:

$$A(T) = \int_0^T \left| \frac{\partial \Phi}{\partial t} \right| dt$$

The activity in an interval $[0, T]$ can be calculated by summing the differences between the adjacent maxima and minima, *i.e.*

$$A(T) = \sum_i |m_{i+1} - m_i| \quad (1)$$

The average activity in an interval $[0, T]$ is given by:

$$\text{AvgActivity}(T) = \frac{A}{T}$$

The following theorem is important because it relates the number of threshold crossings made by a DEVS simulator, activity over a time interval T and the quantum size D .

The number of threshold crossings in an interval of length T for threshold levels that are equally spaced by quantum size, D , is:

$$\text{NumberOfThreshodCross}(T, D) = \frac{A(T)}{D} \quad (2)$$

Equations (1) and (2) hold for continuous curves with a finite number of extrema in an interval - there will be slight error which is bounded by the quantum size, which will disappear as the quantum goes to zero. By definition, Equation (2) will be true for any quantum-based method that takes exactly one quantum step at each transition and tracks the curve exactly. Of course, a method may take more steps than given by Equation (2) due to oscillations that may arise, if the method does not suppress them. Thus Equation (2) gives a lower bound, and indeed, a tight lower bound on the number of transitions that must be taken by a quantum-based method that takes one quantum step at each transition. However, if a quantum-based method takes more than one quantum step at each iteration (as does QSS2 – see [28]) then it can take fewer steps than given by Equation (2). However, in this case, we can conjecture that such methods will be characterized by a maximum number, M , of quantum steps per transition so that its number of transitions will be bounded below by the *activity*/ M . Future work can test this conjecture.

After reviewing solution approaches to our example partial differential equation system, we will return to computing the activity of this system as a basis for comparing the efficiency and accuracy of the solution techniques considered in this paper.

1.2.3 A brief review of basic concepts in implementing quantized integrators

We now review the basic concepts of the activity measure for continuous dynamic system first presented in [7, 14]. Many numerical discrete-time methods are based on discretizing continuous curves of ordinary differential equations (ODEs) using a time step that is applied uniformly to the system components. In contrast to this approach, quantization reduces the number of computations needed to approximate a curve by allocating the computational resources to the active regions. When the value of the curve changes rapidly, the computations are increased to track changes. Otherwise, the number of computations is reduced. Curve changes are related to activity. Mathematically describing activity allows us to predict the number of computations required by a quantization-based method.

A continuous scalar system that is spatially independent can be represented by the ODE:

$$\frac{d\Phi(t)}{dt} = f(\Phi(t)) \quad (3)$$

Where $\Phi(t)$ is a continuous function of time.

Using the Forward Euler method, Equation 3 can be discretized as shown below:

$$\Phi^{n+1} = \Phi^n + \Delta t \cdot f(\Phi^n) \quad (4)$$

Where Δt represents a fixed time step.

Quantization of ordinary differential equations has been introduced in [7, 16, 22, 29]. Rather than calculating the approximated solution at every time step Δt , the quantization method facilitates the computation of the solution by tracking any significant changes in the system. This is possible because in the quantized solution, no processing occurs below a predefined threshold called *quantum*.

In this paper, the basic concepts of the quantized method are reviewed for a single equation. The extension to systems of equation is straightforward; informally, each state variable is updated only when itself or one of its influencers changes by a quantum. This approximation is formalized as a discrete event system and simulated with a discrete event simulator. For a complete description of the technique, see, e.g., [28].

A quantum can be defined as:

$$D = |\Phi^{n+1} - \Phi^n| \quad (5)$$

where D corresponds to the quantum size.

The time required for such a quantum change can be calculated using finite differences in time [*c.f.* Equation (4)]:

$$\Delta t = \frac{D}{|f(\Phi^n)|}$$

(6)

Note that the time step Δt in Equation 6 is variable as it depends on the absolute value of the time derivative $|f(\Phi^n)|$.

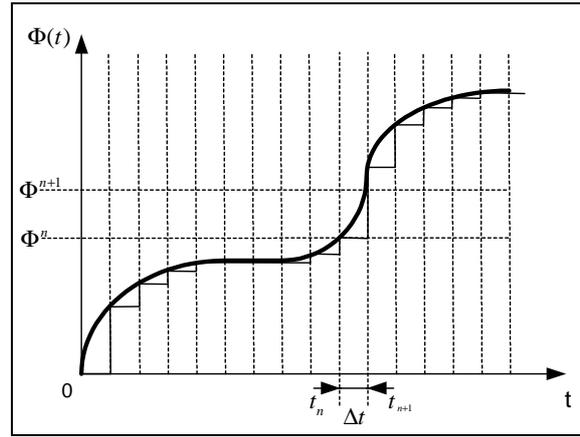
A numerical integration schema can be defined by substituting the value of Δt of Equation (6) in Equation (4). The sign of $f(\Phi^n)$ gives the sign of the derivative $\frac{d\Phi(t)}{dt}$ which is needed to track the direction of the solution. Thus, we obtain a **quantized integrator**:

$$\boxed{\begin{aligned} \Phi^{n+1} &= \Phi^n + D \cdot \text{sign}(f(\Phi^n)) \\ t_{n+1} &= t_n + \frac{D}{|f(\Phi^n)|} \end{aligned}} \quad (7)$$

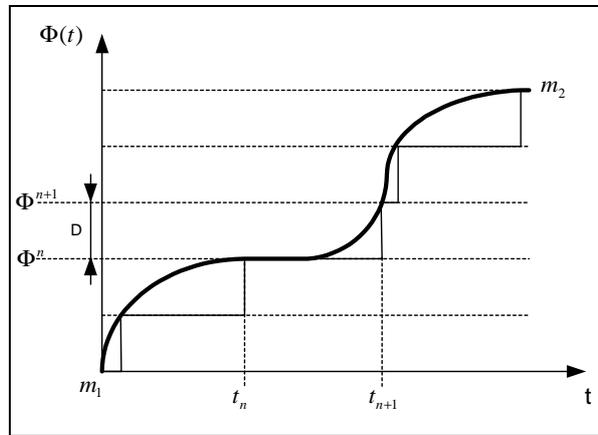
$$(8)$$

where $t_{n+1} \in \mathbb{R}$ corresponds to the time where $f(\Phi^n)$ changes.

Figure 3 depicts two numerical approximation examples: (a) using a discrete-time approach, and (b) using a quantization approach. The function $\Phi(t)$ has a local minimum m_1 and a local maximum m_2 . Notice that, by using a real time base, the number of computations necessary for the approximation of function $\Phi(t)$ can be reduced. This is especially true when the function varies slightly in time.



(a)



(b)

Figure 3. Numerical approximations of an ODE

2 The One-dimensional Diffusion Equation

We use diffusion as an example to compare quantized and discrete-time numerical solutions. After characterizing the analytical solution to a one-dimensional linear diffusion problem, we formulate explicit, implicit and quantized solution techniques for this problem. This section closes with computation of the activity when the initial condition is a Gaussian pulse.

2.1 Definition

The one-dimensional diffusion is represented by equations below:

$$\begin{cases} u_t = cu_{xx} & \text{for } x \in]0, L[\text{ and } t \in]0, T] \\ u(0, t) = u(L, t) = 0 & \text{for } t \in [0, T] \\ u(x, 0) = u_0(x) & \text{for } x \in [0, L] \end{cases} \quad \begin{matrix} (9a) \\ (9b) \\ (9c) \end{matrix}$$

where, in equation (9a), u_t is the first order partial differential of the state variable u with respect to time, c is the diffusion constant and u_{xx} is the second order partial derivative with respect to the space coordinate x , L is the length of the domain, and T is the total time of simulation.

Equation (9b) is the homogeneous Dirichlet boundary conditions. Equation (9c) is the initial condition. Equation (9a) is usually solved by analytical or discrete-time methods. These classical solutions are described in Section 2.2.

2.2 Solutions

The one-dimensional diffusion equation can be solved numerically or analytically. This section presents an analytical solution, two discrete-time numerical solutions (using implicit and explicit schemes) and the quantized numerical solution. The analytical solution is used to compare the numerical solutions.

2.2.1 Analytical Solution

Using the separation of variables method, we obtain

$$u = \sum_{k \in \mathbb{N}} D_k e^{-c\lambda_k^2 t} \sin \lambda_k x \quad (10)$$

where $\lambda_k = \frac{k\pi}{L}$.

Using the pulse initial condition, with dx the variation in space:

$$\begin{cases} u(x, 0) = 1 & \text{for } x \in \left[\frac{L}{2}, \frac{L}{2} + dx \right] \\ u(x, 0) = 0 & \text{elsewhere in the domain} \end{cases}$$

the coefficients D_k are identified as

$$D_k = \frac{2}{L\lambda_k} \left(\cos \lambda_k \left(\frac{L}{2} + dx \right) - \cos \frac{\lambda_k L}{2} \right) \quad (11)$$

Equations (10) and (11) constitute the **analytical solution** of the one-dimensional diffusion equation.

2.2.2 A Discrete-time Solution

The method of lines [30] is a well known general procedure for approximating PDEs to a system of ODEs (see http://www.scholarpedia.org/article/Method_of_lines for more information). Using the forward Euler space method [18], Equation (9a) can be rewritten as

$$u_i^{n+1} = u_i^n + \left(\frac{c\Delta t}{\Delta x^2} \right) \left[u_{i+1}^n - 2u_i^n + u_{i-1}^n \right] \quad (12)$$

where Δx is the mesh size and u_i^{n+1} is the updated value of u for each iteration. This is the **explicit solution** of the one-dimensional diffusion equation.

Using the backward time centered space method, equation (9a) can be rewritten as

$$u_i^{n+1} = \frac{1}{1 + \frac{c\Delta t}{\Delta x^2}} \left[u_i^n + \frac{c\Delta t}{\Delta x^2} (u_{i-1}^{n+1} + u_{i+1}^{n+1}) \right] \quad (13)$$

The solution is calculated at each time step using the iterative method of Jacobi [31] for which a convergence condition is used to pass at next time step. This is the **implicit solution** of the one-dimensional diffusion equation.

Figure 4 depicts the explicit (a) and implicit (b) solutions. At every time step Δt , solutions are computed using at each cell (or grid point) using the states of the cell and its neighbors.

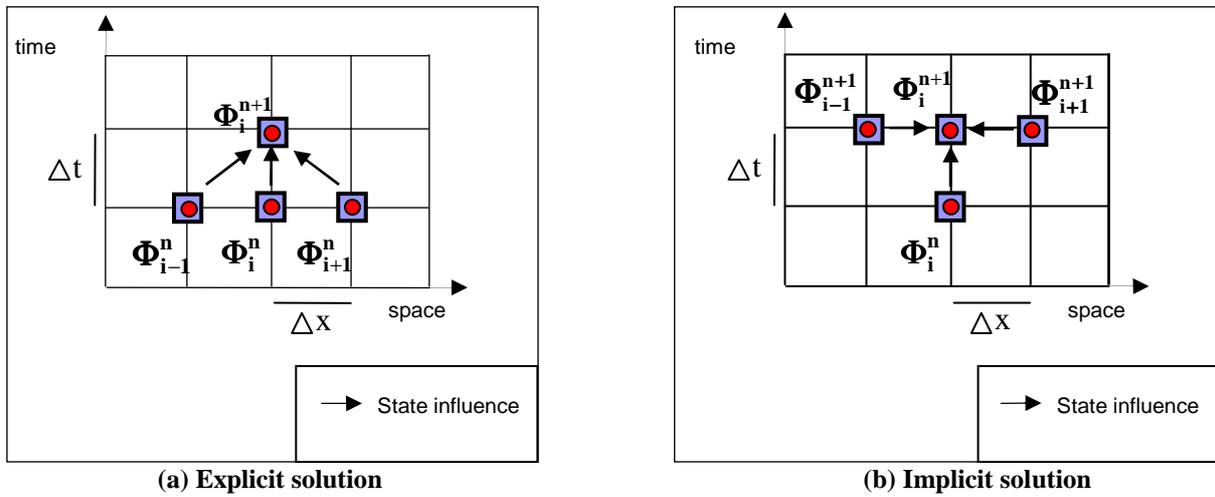


Figure 4. Discrete-time solutions of the one-dimensional diffusion equation

2.2.3 The Quantization Solution

The derivation of this method begins with the discretization of the spatial derivatives [32]. This creates a system of coupled ODEs. The second step consists of choosing a time discretization technique (in this case, explicit Euler). To achieve a quantized discretization, space is still discretized using a classical scheme (in this case, centred differences), but then the quantized integrators described by Equations (7) and (8) are used to discretize the state space. Hence, by first discretizing the spatial dimension of Equation (9a) using centered finite differences, this gives a system of ordinary differential equations described by:

$$\frac{du_i}{dt} = \left(\frac{c}{\Delta x^2} \right) [u_{i+1} - 2u_i + u_{i-1}] \quad (14)$$

The resulting system of ordinary differential equations is then solved using quantized integrators. Every integrator (or cell) can be specified as a DEVS atomic model integrator (the semantics of DEVS atomic models is described in Appendix). This gives a discrete event approximation of the diffusion process for which new states are computed when the solution at any spatial grid point changes significantly [5, 7].

Figure 5 depicts the quantization solution of the one-dimensional diffusion equation. Notice that the cell's next state is computed using the cell's previous state (updated at the last neighbor's boundary crossing), and the

last state of its neighbors, which have been communicated at quantum boundary crossings. A discrete-time base Δt has been represented for comparison.

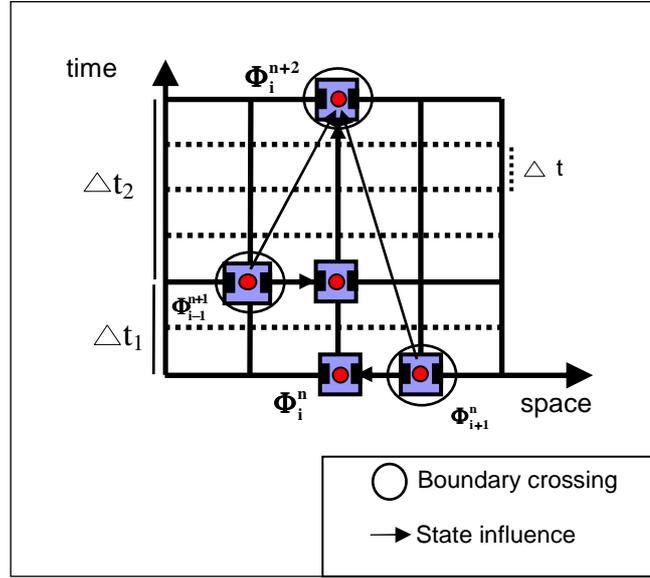


Figure 5. Quantization solution of the one-dimensional diffusion equation

2.3 Activity for a Gaussian pulse as the Initial Data

We consider the case of a Gaussian pulse as the initial data for the case of open boundary conditions where the heat of the system tends to zero. It is not possible to start a simulation with time equal to zero and using a Gaussian pulse as initial condition because the pulse becomes a Dirac delta function concentrated at the origin as t approaches zero. So in the following analysis, we start from a state in which time starts from t_{start} , some time greater than zero.

As described in [14], using Equation (1) and the error function $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-z^2} dz$ we obtain:

$$A \approx \frac{NH \cdot f(L)}{L}$$

where,

N is the number of cells,

H is the amplitude of the Gaussian pulse, and

$$f(L) = \left(\frac{2}{\sqrt{2\pi e}} \ln \left(\frac{L}{\sqrt{2ct_{start}}} \right) - \frac{1}{2} erf \left(\frac{L}{\sqrt{4ct_{start}}} \right) + erf(0.707) \right) \quad (15)$$

We note that as N gets large, the average activity $\frac{A}{N}$ approaches the constant $\frac{H f(L)}{L}$, the total activity A increases linearly with N , the number of cells, and $\frac{A}{N}$ decreases as L increases since the error function goes to a constant and the natural logarithm term grows more slowly than L .

We use three different discretizations of space to study the benefits of using quantization to simulate the diffusion process. These configurations are

- **Constant mesh configuration:** Different domain lengths and a constant space step $\Delta x (= L/N)$,
- **Mesh enlargement configuration:** Different domain lengths and a constant number of cells,
- **Mesh refinement configuration:** A constant domain length and an increasing number of cells.

Figure 6 shows the results of using equation (13) to compute the total activity as a function of the number of cells when the density of cells is kept constant. As can be seen, there is a slight increase in activity as the number of cells increases. From equation (13) we can see that the activity of the Gaussian pulse is given by

$$A \approx \frac{NH}{L} f(L) = \frac{H}{\Delta x} f(L)$$

Therefore, the activity increases very slightly with increasing L due to the dominance of the logarithmic term in the composition of $f(L)$. This behavior of total activity is experimentally verified in the simulation results in the next section (see Figure 14).

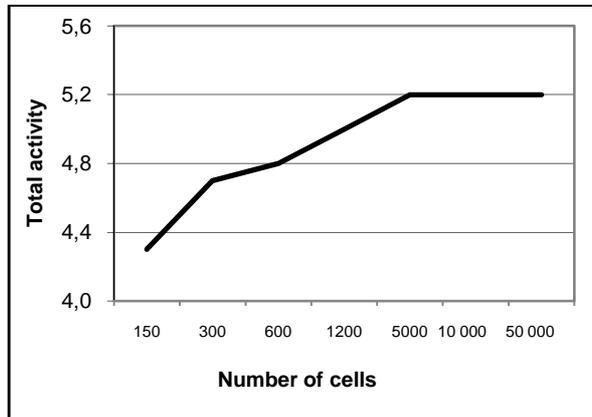


Figure 6. Activity versus number of cells when Δx is constant

Figure 7 shows the total activity as function of the length of the computational domain. We see that increasing the length in this manner decreases the activity. From Equation (15) we can see that the activity of the Gaussian pulse decreases due to the dominance of the linear term in L over the other terms. This behavior of total activity is experimentally verified in the simulation results in Sub-section 3.3 (see Figure 14).

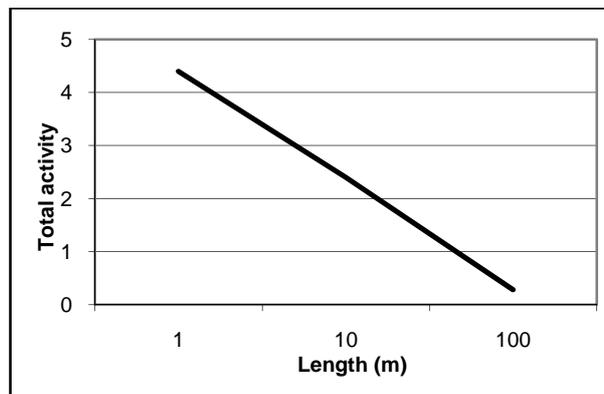


Figure 7. Activity versus length for a constant number of cells

From Equation (15), we can see that as the number of cells, N keeps increasing while the length of the cell space, L is kept constant, the activity increases in direct proportion to N . Since a method based on Quantization keeps track of activity, the order of complexity of a quantized method will be $O(N)$. The discrete-time methods considered in this paper do not make use of the heterogeneity in activity and hence the order of complexity of these discrete-time methods is $O(N^2)$. This behavior of total activity is experimentally verified in the simulation results in the next section (see Figure 19).

3 Simulation Results

In this section, we compare the results of simulations using the implicit, explicit, and discrete event methods described in Section 22.2. Implementations have been achieved using the C++ programming language and the ADEVS simulator (accessible online: <http://www.ornl.gov/~1qn/adevs/index.html>.) Examining these results confirms the activity theory, as well as the computational advantage of the quantized solution over the discrete-time based numerical solution methods.

We consider the simulation of heat diffusion for a domain with homogeneous Dirichlet boundary conditions. The simulation is run for $8s$ with the initial condition as a pulse. The value of the diffusion constant is $c = 0.01$. Errors of the numerical solutions are determined by comparison with the known analytical solution. Simulation results of the quantized solution are compared to the explicit and implicit ones. Results are analyzed and explained using the activity theory. The simulations were performed using a *1.5 GHz Pentium M* processor.

In the following we depict the execution times, number of transitions and average errors for (1) a constant rod length, a constant number of cells and different quantum sizes, (2) different number of constant size cells and different rod lengths, (3) a constant number of cells and different rod lengths, and (4) an increasing number of cells and a constant rod length.

3.1 Determination of the Quantum Size

The temperature at the left and right ends is kept at zero. At time $t = 0s$, a pulse of amplitude 1 is applied at the center of the domain. The length of the domain is equal to $1m$ and the grid width $\Delta x = 0.01m$. The domain is divided into $N = 101$ cells. Figure 8 depicts the 3D activity distribution.

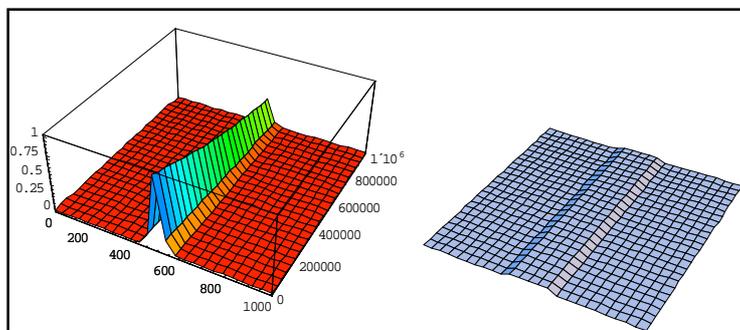


Figure 8. 3D activity distribution

Figure 9 depicts the analytical solution. The diffusion of the pulse is shown at times $t = 0.2s$, $t = 0.8s$ and $t = 6s$. At time $t_{end} = 8s$, the heat is totally diffused.

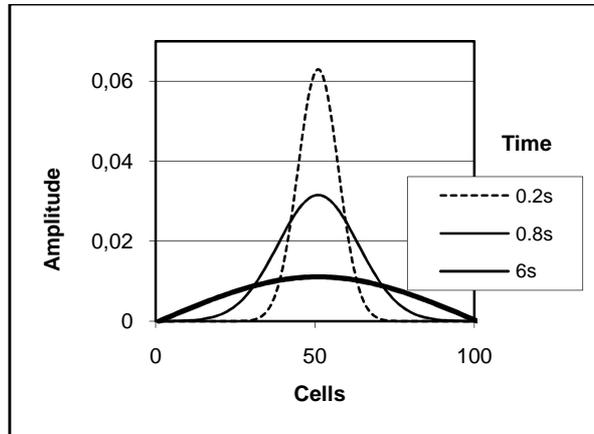


Figure 9. Analytical solution

A first choice of quantum size consists of determining the quantum size to avoid oscillations at the equilibrium. If the equilibrium solution is known, then the quantum size can be selected so that equilibrium points are reachable in the discrete state space [7]. This ensures that, rather than oscillating about the equilibrium (see, *e.g.*, [33]) the numerical solution will settle at a point where the time advance is infinite and the simulation does no unnecessary work. The specific requirement for this to occur is that the difference between the initial state and final state at each grid point be an integer multiple of the quantum. Figure 10 compares the results of choosing an incorrect ($0.9e-7$) and correct ($1e-6$) quantum size.

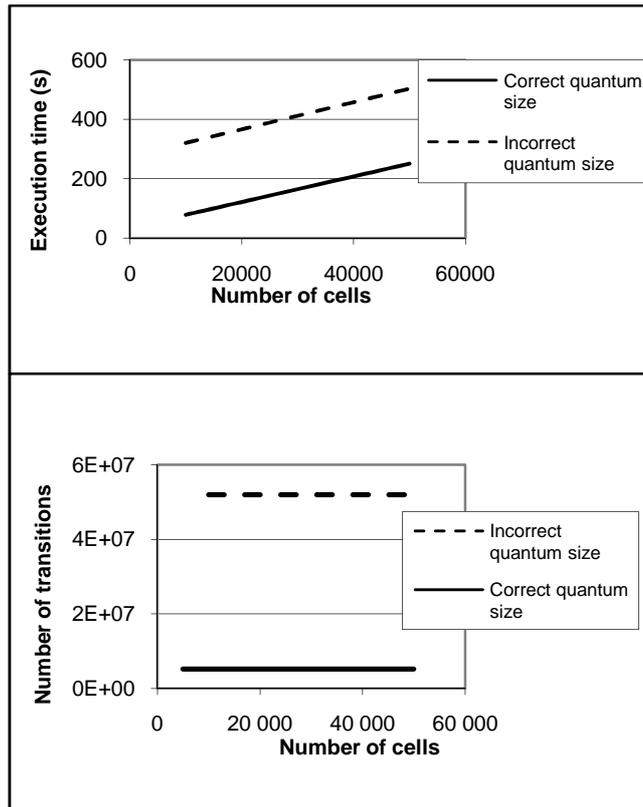


Figure 10. Incorrect and correct choices of quantum size scales

After choosing the right quantum size scale, the final quantum size needs to be determined. Figure 11 depicts errors obtained when increasing the quantum size.

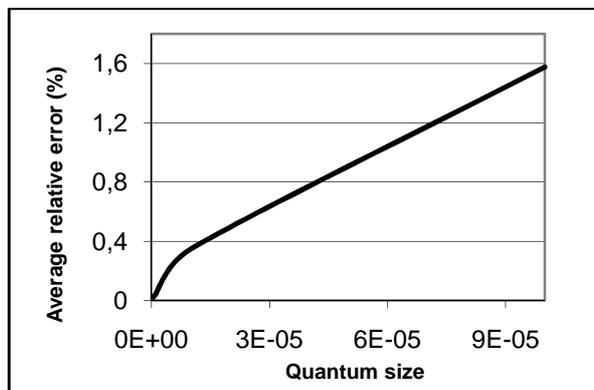


Figure 11. Average relative error according to quantum size

In the same way, increasing the time step of the explicit method leads to error increase as depicted in Figure 12.

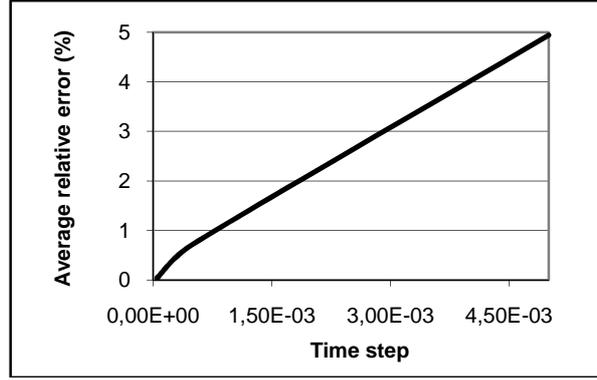


Figure 12. Average relative error according to time steps

Table 1 depicts the simulation results for different quantum sizes. We see that the bigger the quantum size, the smaller the number of transitions and the execution time. Concomitantly, the smallest time advance (ta_{min}) in the simulation increases with increasing quantum size. This time corresponds to the minimum time for the system's state to change by a quantum. For quantum sizes that are smaller than 10^{-5} , Equation (2) (stating inverse dependence of transitions on quantum size) is verified. That is, reducing the quantum size by a factor of 10 increases the number of transitions by 10.

Quantum size	# of transitions	Execution time(s)	$ta_{min}(s)$
10^{-7}	$4.5 \cdot 10^7$	221	$5 \cdot 10^{-10}$
10^{-6}	$4.5 \cdot 10^6$	22	$5 \cdot 10^{-9}$
10^{-5}	463 693	2	$5 \cdot 10^{-8}$
10^{-4}	91 541	1	$5 \cdot 10^{-7}$
10^{-3}	60 489	1	$5 \cdot 10^{-6}$

Table 1. Simulation results for different quantum sizes

To compare the errors of the numerical solutions against the analytical one, we use the following definition of average relative error:

$$\bar{\varepsilon} = \sum_{i=1}^N \frac{1 - |q_i / q_i^*|}{N}$$

where N = number of cells,

q_i = analytical value of cell i ,

q_i^* = numerical value of cell i .

In the following, we use a quantum size $D = 10^{-6}$ and a time step $\Delta t = 5 \cdot 10^{-5} s$ for the implicit and explicit methods. Due to these very small time steps, the average relative errors of the different methods are very small and almost the same. Considering the total average relative error, $\bar{E} = \sum_{t=0}^{t_{end}} \bar{\varepsilon}(t)$ in each case is

$\bar{E}_{\text{quantization}} = 0.040\%$, $\bar{E}_{\text{explicit}} = 0.036\%$ and $\bar{E}_{\text{implicit}} = 0.087\%$. Moreover, since the diffusion is completed after 8s of propagation, keeping the same grid cell size Δx , errors will remain the same for greater domain lengths.

The purpose of our comparison is to show that the total error - the combined error of both time and space discretization - of the two schemes (Forward or Backward Euler or Quantization) is essentially the same. In practice, the total error is the only error that matters. If the error due to the spatial discretization dominates, then the choice of integrator in time should not be driven by concerns over accuracy: computational efficiency is a more important consideration in this case. Our comparison illuminates this point.

3.2 Constant Mesh Configuration

Here, for the same cell size ($\Delta x = 0.01m$), the number of cells has been increased. Figure 13 compares execution times of the numerical methods using this fact. The execution times of the quantized method are lower than the execution times of the discrete time numerical methods. This is due to the ability of the quantized simulation to focus computation resources on the high activity zones.

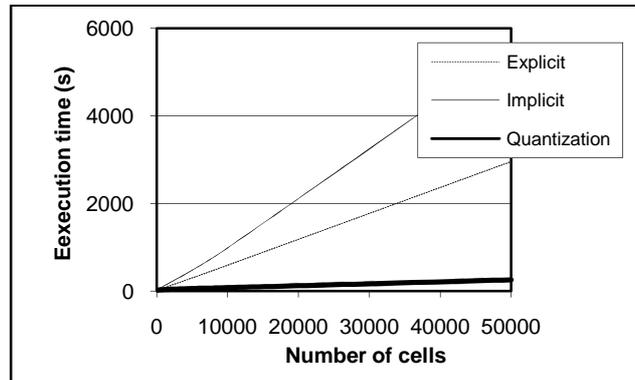


Figure 13. Execution times for a constant cell's size and increasing numbers of cells

Indeed, Figure 14 pinpoints the activity tracking capability of the quantization method. The number of transitions of the discrete-time methods is strongly increasing with the number of cells. As the quantization method just concentrates on active cells, the number of transitions almost remains constant. This corresponds to the total activity as discussed in Section 2.4. Note the correspondence between Figures 6 and 14 that are related through Equation (15). The number of transitions made by the quantization method in Figure 14 as the number of cells increases is directly related to that of the total activity in Figure 6.

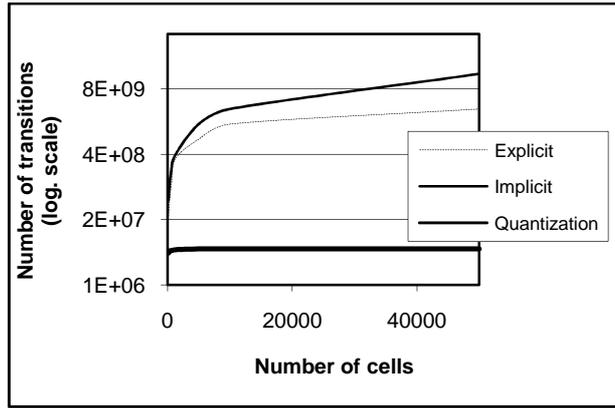


Figure 14. Number of transitions for a constant cell's size and increasing numbers of cells

As the size of the cell is constant and the diffusion, which is mainly concentrated one meter around the pulse, is finished at the end of the simulation, the average relative error of Section 3.1 remains valid here.

3.3 Mesh Enlargement Configuration

Figure 15 depicts the execution times obtained when keeping the same number of cells $N = 100$, we increase the domain length. Implicit and explicit methods give execution times of $13s$, whatever the domain length. Concerning the quantization method, execution times decrease from $40s$ (for $L = 1m$) to $23s$ (for $L = 10m$), and finally to $3s$ (for $L = 100m$).

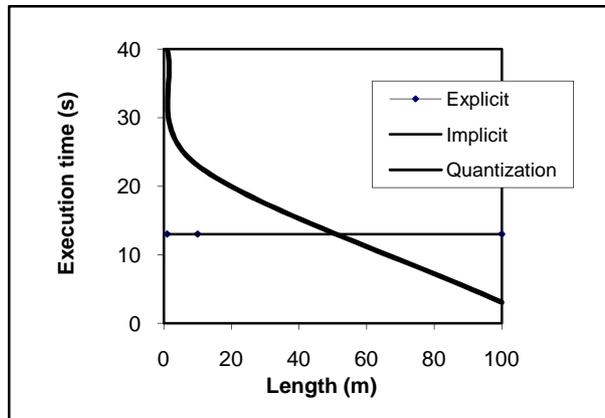


Figure 15. Execution times for a constant number of cells and different rod lengths

Increasing the space discretization, we obtain a linear function for the total mean error which is the same for explicit and quantization methods:

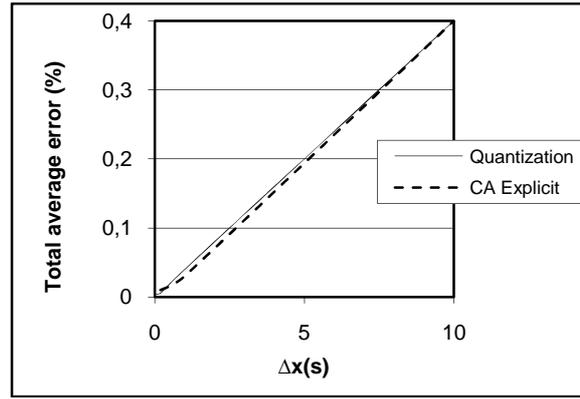


Figure 16. Total average error according to space discretization

Figure 17 represents the number of transitions obtained for different methods, a constant number of cells and different rod lengths. Though the number of transitions of implicit and explicit methods is constant, the number of transitions of the quantization method decreases as the length increases. This is because the time advance (8) of the quantization method is calculated using the centered differences in space (14). The more the space step Δx increases, the more the time advance increases.

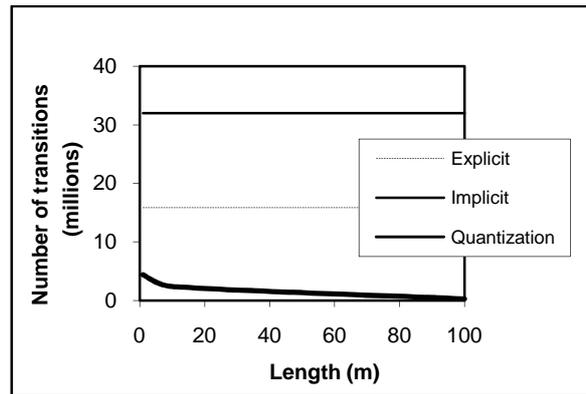


Figure 17. Number of transitions for a constant number of cells and different rod lengths

Notice that the number of transitions of the quantization method is significantly lower than the ones for the implicit and explicit methods. However, execution times of the quantization method can be greater for $L \leq 50m$. This is due to an overhead induced by the algorithm used to focus on activity. The quantization method is more interesting if the activity zone is small with respect to the diffusion domain. This corresponds to the behavior of total activity as discussed in the previous section. Note the correspondence between Figures 7 and 17.

The average relative errors of the explicit and quantization methods for $L = 10m$ and $N = 100$. Errors are identical. Total average relative errors are very small (still due to small time-steps and quantum size): $\bar{E}_{quantization} = 0.42\%$, $\bar{E}_{explicit} = 0.40\%$, and $\bar{E}_{implicit} = 0.40\%$.

3.4 Mesh Refinement Configuration

Figure 18.b shows the execution times of the different methods for $L = 1m$ and an increasing number of cells. This is a zoom on the beginning of the x-axis of Figure 18.a. Diffusion occurs on the whole length $L = 1m$. The number of cells is increased in this zone of high activity. Until $N = 350$, execution times of the quantization method are slightly greater than the discrete-time methods. This is due to the fact that activity detection of the quantization method takes more time than simply calculating on the whole domain. Figure 19 confirms this guess. In this figure one can notice that the number of transitions of the quantization method is smaller than the number of transitions of the discrete-time methods.

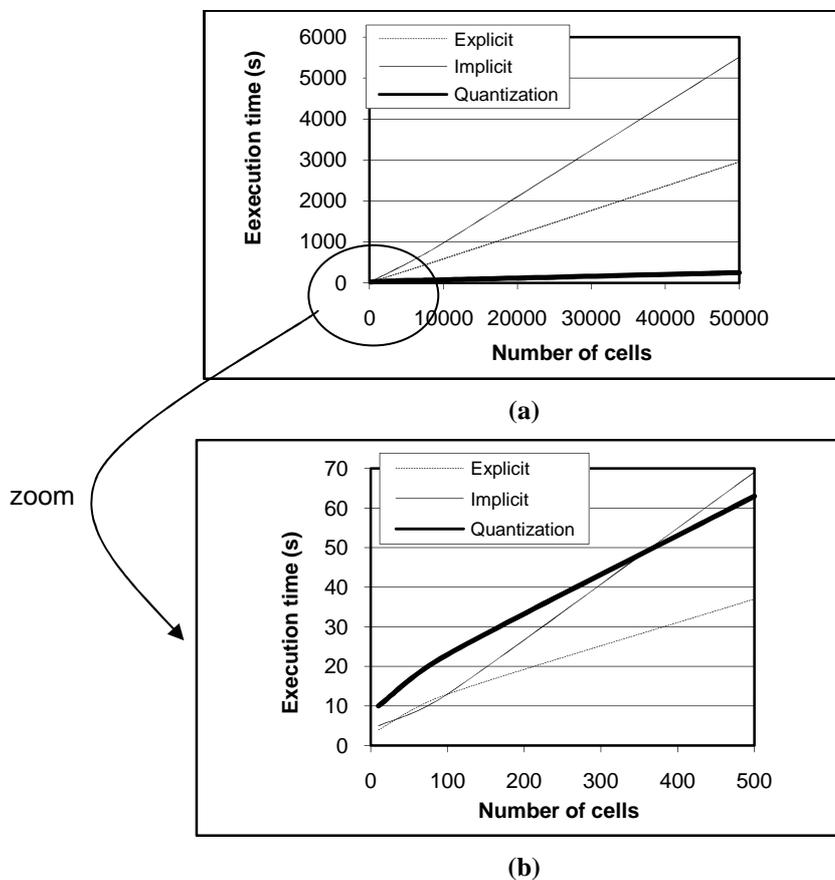


Figure 18. Execution times for an increasing number of cells and for $L = 1m$

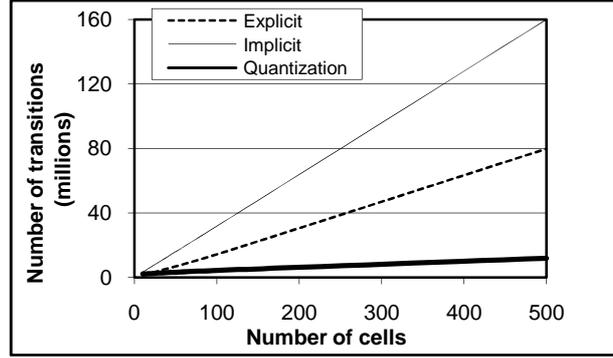


Figure 19. Number of transitions for an increasing number of cells and for $L = 1m$

Recall that Equation (15) predicted that as the number of cells N increases, while the length of the cell space is kept constant, the activity increases in direct proportion to N . Figure 19 confirms this prediction as well as the faster increase in transitions for the discrete-time methods.

Average relative errors for $L = 1m$ and $N = 500$ cells are still very small: $\bar{E}_{quantization} = 0,006\%$, $\bar{E}_{explicit} = 0,013\%$, and $\bar{E}_{implicit} = 0,01\%$. This is due to a very small space discretization: $\Delta x = 0.002m$. Hence, the solution is very accurate and errors can be considered as numerical oscillations because of their small amplitude and small duration ($0.4s$).

4 Conclusion

Tracking activity in space has been shown to be more advantageous than computing non-active components even allowing for the extra overhead it may incur. A comprehensive activity-based modeling and simulation framework has been presented and applied. Furthermore, we have introduced a first solution to achieve activity tracking using discrete-event modeling and simulation. Basic Euler methods have been used to illustrate this new activity paradigm approach. Although these discretization methods are known to be computation-intensive¹, they allow us to more easily introduce many formal, computational concepts and compare them with conventional methods. The efficiency, stability, and accuracy of the method examined in this paper, and of several related methods, have been studied in [8-13]; these papers discuss extensions of the quantization concept to stiff systems and to higher order numerical integration methods. Although this paper looked only at a first order accurate integration scheme, the same principles are naturally extended to higher order methods [3-6, 9-12] and specialized methods for stiff and differential algebraic systems [10, 11, 34].

The relative improvement of the quantized integration schemes over the discrete time Euler methods is due, in part, to the inability of these methods to track activity. Such methods can be enhanced to perform such tracking to good effect [3, 35]. However, more fundamentally, the quantum-based schemes support a restatement of the simulation problem whereas discrete time schemes ask the question “given the solution at time t , what will

¹ Notice that backward Euler or any backward differentiation formulas method can use larger time steps than the small time steps used in this study (thus leading to better execution times.) Conversely, using larger quanta will lead to oscillations because of the stiffness of the system [36]. However, the scope of this paper was to establish a new activity-based framework, under the same experimental conditions, and to apply it for the comparison of discrete-event and discrete-time discretization techniques.

be the solution be at time $t+h$ ", discrete event methods ask "given the solution now, when will the solution change by a quantum-sized amount". This new simulation paradigm is made operational by approximating continuous systems using a discrete state space and a continuous time base. In contrast to this, discrete time methods employ a discrete time base and continuous state space.

When quantized integrators are implemented as discrete event systems, the resulting simulation algorithm automatically and generically tracks activity in space. This is a natural result of the discrete event simulation algorithm, and so it does not require special procedures to track changes in a cell's states [21, 35]. The activity measure allows us to predict the efficiency of the quantization simulations of continuous systems. However, to calculate activity in advance of actual simulation requires analytical solution of the continuous system. Generally, complex PDEs do not have analytical solutions so that smart methods must be developed to monitor and extrapolate the current activity, in order to exploit activity theory's prediction capability. In any case, the theory suggests that systems with temporal and spatial variation in activity are good candidates for activity-based techniques. The results of some recent experimental works have been reported (see [5, 35]) with favorable results.

Further experimentation is needed, and more complex systems have to be studied, to assess efficiency of quantization and other activity-based techniques in a broader context. Also comparison with other recent methods such as those of Logg's [25] and Takahashi's [26] will have to be done in future research to establish the relationship of discrete event techniques to methods that have evolved from traditional approaches. Nonetheless, we have demonstrated that the concept of activity can estimate the computational effort required by a quantized state method. Furthermore, our experiments have shown that when properly designed, a discrete event simulator for such a method achieves a reduction in the number of state transitions that more than compensates for the overhead it imposes.

References

1. Muzy, A. and B.P. Zeigler, *Introduction to the Activity Tracking Paradigm in Component-Based Simulation*. The Open Cybernetics and Systemics Journal, 2008. **2**: p. 48-56.
2. Jammalamadaka, R., *Multilevel Methodology for Simulation of Spatio-Temporal Systems with Heterogeneous Activity: Application to Spread of Valley Fever Fungus*. 2008, University of Arizona: Tucson.
3. Muzy, A., et al. *Efficient simulation of large-scale dynamic structure cell spaces*. in *Summer Computer Simulation Conference*. 2003. Montréal, Canada: SCS.
4. Muzy, A. and J.J. Nutaro. *Algorithms for efficient implementation of the DEVS & DSDEVS abstract simulators*. in *1st Open International Conference on Modeling and Simulation (OICMS)*. 2005. Clermont-Ferrand, France.
5. Muzy, A., et al. *Dynamic structure cellular automata in a fire spreading application*. in *1st International Conference on Informatics in Control Automation and Robotics (ICINCO 2004)*. 2004. Setubal, PORTUGAL.
6. Nutaro, J., *A discrete event method for wave simulation*. *Acm Transactions on Modeling and Computer Simulation*, 2006. **16**(2): p. 174-195.
7. Nutaro, J., et al. *Discrete event solution of gas dynamics within the DEVS framework*. in *International Conference on Computational Science (ICCS 2003)*. 2003. Melbourne, Australia.
8. Nutaro, J. and B. Zeigler, *On the stability and performance of discrete event methods for simulating continuous systems*. *Journal of Computational Physics*, 2007. **227**(1): p. 797-819.
9. Kofman, E., *A Second Order Approximation for DEVS Simulation of Continuous Systems*. *Simulation (Journal of The Society for Computer Simulation International)*, 2002. **78**(2): p. 76-89.

10. Migoni, G. and E. Kofman. *Linearly Implicit Discrete Event Methods for Stiff ODEs. Part I: Theory*. in *RPIC 2007*. 2007.
11. Migoni, G. and E. Kofman. *Linearly Implicit Discrete Event Methods for Stiff ODEs. Part II: Implementation*. in *RPIC 2007*.
12. Nutaro, J. A *Second Order Accurate Adams-Bashforth Type Discrete Event Integration Scheme*. in *21st International Workshop on Principles of Advanced and Distributed Simulation (PADS '07)*. 2007.
13. Kofman, E. and S. Junco, *Quantized State Systems. A DEVS Approach for Continuous Systems Simulation*. Transactions of SCS, 2001. **18**(3): p. 123-132.
14. Muzy, A., et al. *Discrete event simulation of large-scale spatial continuous systems*. in *IEEE International Conference on Systems, Man and Cybernetics*. 2005. Waikoloa, HI.
15. Balci, O. *The implementation of four conceptual frameworks for simulation modeling in high-level languages*. in *Winter Simulation Conference*. 1988.
16. Zeigler, B.P., H. Praehofer, and T.G. Kim, *Theory of modelling and simulation*. 2000: Academic Press.
17. Muzy, A., et al., *Modeling and simulation of fire spreading through the activity tracking paradigm*. Ecological Modelling, 2008. **219**(1): p. 212-225.
18. Ralston, A. and P. Rabinowitz, *A First Course in Numerical Analysis*. 2001: Dover Publications.
19. Zeigler, B.P., *Discrete event abstraction: an emerging paradigm for modeling complex adaptative systems*, in *Adaptation and evolution (festschrift for John H. Holland)*, E. Oxford press, Editor. 2005, Santa Fe Institute.
20. Muzy, A. and X. Hu. *Specification of Dynamic Structure Cellular Automata & Agents*. in *IEEE Melecon'08*. 2008. Ajaccio, France.
21. Santoni, P.A., *Elaboration of an Evolving Calculation Domain for the Resolution of a Fire Spread Model*. Numerical Heat Transfer, Part A: Applications, 1998. **33**(3): p. pp. 279-298.
22. Kofman, E., J.S. Lee, and B.P. Zeigler. *DEVS representation of differential equation systems: Review of recent advances*. in *13th European Simulation Symposium*. 2001. Marseille, France.
23. LeVeque, R.J., *Numerical Methods for Conservation Laws*, ed. B. Verlag. 1996.
24. Yu, H., *A Local Space-Time Adaptive Scheme in Solving Two-Dimensional Parabolic Problems based on Domain Decomposition Methods*. Siam J. Sci Comput., 2001. **23**(1): p. 304-322.
25. Logg, A., *Multi-adaptive time integration*. Special issue: Workshop on innovative time integrators for PDEs. Applied Numerical Mathematics, 2004. **48**(3-4): p. 339 - 354.
26. Takahashi, K., et al., *A multi-algorithm, multi-timescale method for cell simulation*. Bioinformatics, 2004. **20**(4): p. 538-546.
27. Jammalamadaka, R., *Activity Characterization of Spatial Models: Application to the Discrete Event Solution of Partial Differential Equations*, in *Electrical and Computer Engineering*. 2003, University of Arizona.
28. Kofman, E., J.S. Lee, and B.P. Zeigler. *DEVS Representation of Differential Equation Systems: Review of Recent Advances*. in *2001 European Simulation Symposium*. 2001. Marseille, France.
29. Bolduc, J.S. and H. Vangheluwe. *Expressing ODE models as DEVS: Quantization approaches*. in *AI, Simulation and Planning in High Autonomy Systems*, SCS. 2002. Lisboa, Portugal.
30. Forsythe, G.E. and W.R. Wasow, *Finite-Difference Methods for Partial Differential Equations*. 1960, New York, NY: John Wiley and Sons INC.
31. Sibony, M. and J.C. Mardon, *Approximations et équations différentielles*. 1988: Hermann.
32. Scheisser, W.E., *The Numerical Method of Lines*. Academic Press ed. 1991, San Diego.
33. Nutaro, J., *Parallel discrete event simulation with applications to continuous systems*. 2003, University of Arizona: Tucson.
34. Kofman, E., *Quantization-Based Simulation of Differential Algebraic Equation Systems*. SIMULATION, 2003. **79**(7): p. 363-376.
35. Jammalamadaka, R., B.P. Zeigler, and I.C. Society. *A generic pattern for modifying traditional PDE solvers to exploit heterogeneity in asynchronous Behavior*. in *21st International Workshop on Principles of Advanced and Distributed Simulation (PADS 2007)*. 2007. San Diego, CA.
36. Migoni, G. and Kofman, E. "Linearly Implicit Discrete Event Methods for Stiff ODEs.". Latin American Applied Research (LAAR Journal), 2009. In press.

Appendix

A Brief Description of DEVS

The Discrete Event System Specification (DEVS) is a modeling framework that describes a systems with two types of modeling structures. State space models (also called atomic models) describe the smallest elements of a system in terms of inputs and outputs, state variables, state transition functions, a time advance function, and an output function. Network (also called coupled or multi-component) models describe how atomic and other network models are interconnected to form larger systems. Network models are described by their inputs and outputs, component model set, and interconnections between component models.

The system theoretic roots of DEVS are apparent in the fact that every DEVS model can be reduced to a classical state space representation in the form $\langle X, Y, S, T, A, \lambda, \Omega \rangle$. The elements X and Y are the system input and output sets, S is the system state set, T is the system time base, A is the system state transition function, λ is the system output function, and Ω is the set of admissible input trajectories. A trajectory is a function from the system time base to the system input set (an input trajectory) or system output set (an output trajectory).

The essential operation described by the structure is the mapping of an input trajectory and initial state to a final state (described by the state transition function) and output trajectory (described by the output function). Specializations of this basic structure are constructed to accommodate specific classes of admissible trajectories. A given specialization is a system if a procedure exists for reducing the specialized structure to the basic system structure described above.

DEVS specializes the basic system structure to accommodate event trajectories. An event trajectory is a function from the real numbers to a set that takes on the special non-event value everywhere except at a finite number of points in any finite interval of time. Figure 1 illustrates this with an event trajectory $x(t)$.

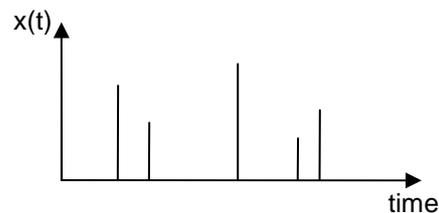


Figure A.1. An example of an event trajectory

A DEVS atomic model is described by a structure $\langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, t_a \rangle$. The sets X , Y , and S are the input, output, and state sets, respectively. The functions δ_{int} , δ_{ext} , and δ_{con} are the internal, external, and confluent state transition functions, respectively. The internal transition function describes the autonomous behavior of the system. The external transition function describes the input response of the system. The confluent transition function describes the evolution of the system state when an internal and external event coincide. The function λ is the output function, and t_a is the time advance function. The time advance function is used to schedule output and internal events.

The abstract simulator for a DEVS atomic model is prescribed by the procedure that is used to reduce its structure to a classical state space representation. While the formalized procedure is described recursively, there exist numerous iterative and parallel implementations of this procedure that are suitable for practical

computations.

One such algorithm for simulating a single atomic model is shown below. Here, tN denotes the next event time, tL is the last event time, t_0 is the time at which the simulation starts, t_f is the simulation stop time, $x(t)$ denotes the input trajectory, $y(t)$ denotes the output trajectory, and q is the system state. The trajectory $y(t)$ initially takes on the value of the null event, denoted by Φ , at all times. The input trajectory $x(t)$ is given prior to simulation start and will be consumed as the computation progresses. The initial system state is denoted by q_0 .

```

t ← 0
for all i ∈ [1; n] do
    tLi ← 0
    set si to the initial state of Mi
end for
while terminating condition not met do
    for all i ∈ [1; n] do
        tNi ← tLi + tai(si)
        Empty the bag xi
    end for
    t ← min {tNi}
    for all i ∈ [1; n] do
        if tNi = t then
            yi ← λi(si)
            for all j ∈ [1; n] & j ≠ i & zij(yi) ≠ Φ do
                Add zij(yi) to the bag xj
            end for
        end if
    end for
    for all i ∈ [1; n] do
        if tNi = t & xi is empty then
            Si ← δint,i(Si)
            tLi ← t
        else if tNi = t & xi is not empty then
            Si ← δcon,i(Si; xi)
            tLi ← t
        else if tNi ≠ t & xi is not empty then
            Si ← δext,i(Si; t ; tLi; xi)
            tLi ← t
        end if
    end for
end for
end while

```

Algorithm A.1. An iterative procedure for simulating a DEVS atomic model.

An example will illustrate the essential concepts. Other, more traditional, examples (e.g., of queuing systems expressed in DEVS) can be found on the web and throughout the literature. Consider a system described by the function $dq/dt = c(t)$, where $c(t)$ is the piecewise constant trajectory shown in Figure A.2. This can be readily encoded as an event trajectory by recording the points at which the amplitude of $c(t)$ changes. This encoding is shown in Figure A.3.

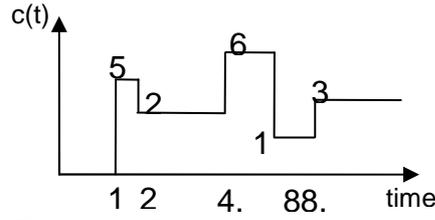


Figure A.2. Piecewise constant trajectory $c(t)$

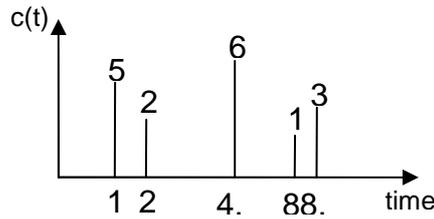


Figure A.3. $c(t)$ encoded as an event trajectory

We can construct a DEVS model that will compute $q(t)$ for any such piece-wise constant $c(t)$. The model will provide outputs at a fixed resolution D in the state space of the system. Let $X = Y = R$ and $S = \{ (q, q', s) / q, q', s \in R \}$, where R is the real numbers. Let D be the desired minimum resolution in the output of the integrator. The state transition functions are given by:

$$\begin{aligned}\delta_{\text{int}}((q, q', s)) &= (q + sq', q', D/|q'|) \\ \delta_{\text{ext}}((q, q', s), e, x) &= (q + eq', x, 0) \\ \delta_{\text{con}}((q, q', s), x) &= (q + sq', x, D/|x|)\end{aligned}$$

The model output and time advance functions are given by:

$$\begin{aligned}\lambda((q, q', s)) &= q + sq' \\ \text{ta}((q, q', s)) &= s\end{aligned}$$

The initial state of the model is $(x(0), 0, \infty)$. Using algorithm A.1 with $D = 1$, the state and output trajectories of the system over the time interval $[0, 4.3]$ can be computed as shown in the Table A.1. Empty entries in the table denote a non-event.

Similar, though more complex, procedures exist for computing the behavior of multi-component and hierarchical models. In practice, these procedures are implemented as part of a simulation package that can be used to implement DEVS models. The simulation package provides abstract classes with virtual state transition, initialization, output, and time advance functions. These classes are implemented by end users and plugged into the simulation engine for execution.

<i>t</i>	<i>state</i>	<i>x(t)</i>	<i>y(t)</i>	<i>ta</i>
0	0, 0, ∞			∞
1	0, 5, 0	5		0
1	0, 5, 0.2		0	0.2
1.2	1, 5, 0.2		1	0.2
1.4	2, 5, 0.2		2	0.2
1.6	3, 5, 0.2		3	0.2
1.8	4, 5, 0.2		4	0.2
2	5, 2, 0.5	2	5	0.5
2.5	6, 2, 0.5		6	0.5
3	7, 2, 0.5		7	0.5
3.5	8, 2, 0.5		8	0.5
4	9, 2, 0.5		9	0.5
4.1	9.2, 6, 0	6		0
4.1	9.2, 6, 0.17		9.2	0.17
4.27	10.2, 6, 0.17		10.2	0.17

Table A.1. State and output trajectories