Introduction to the Activity Tracking Paradigm in Component-Based Simulation

A. MUZY*

Laboratory CNRS LISA, Università di Corsica – Pasquale Paoli, UFR Drittu, Scenzi suciali, ecunòmichi è di gestioni, 22, av. Jean Nicoli, BP 52, 20250 Corti, France.

B.P. ZEIGLER

Arizona Center for Integrative Modeling and Simulation, Department of Electrical and Computer Engineering, University of Arizona, 1230 E Speedway Blvd, Tucson, Arizona, USA.

Abstract: Dynamic systems are studied and modeled through the description of state changes (or trajectories). State changes over time constitute the activity of a dynamic system. In a component-based simulation, state changes depend on both computations and exchanges of information. In this paper, the activity tracking paradigm is introduced as a guide for modeling dynamic systems and developing corresponding efficient component-based simulations. Mechanisms and structures to track and describe activity through information are informally and formally introduced.

INTRODUCTION

Currently, Simulation opens new perspectives to Science. In contrast to the era of hand computation, today increasingly the dynamics of complex systems are directly modeled to be simulated on digital computers. This induces a shift of methodology in the scientific community. Modeling theories are selected to fit the ability of computers to deal automatically with digital information.

Figure 1 is a 3D picture of a fire spreading simulation. It is a good introduction of what is a simulation goal in "virtual reality." To achieve this goal (modeling a fire spread), consider two alternative questions: (i) "You are a point in space at position *(x,y,z)*, are you in fire? Will you receive heat energy?" (asking all non-burning points in space), or (ii) "You are a burning point in space at position *(x,y,z)*, how will you propagate heat energy?" Answering to the second question (which is more intuitive and turns out to be more efficient) leads to the approach of activity tracking and specification, in space and time.

When dealing with dynamic complex systems, the scope is to describe activity and topology of systems. In simulation, complexity of systems depends on: (i) the quantity of digital information to store, (ii) the quantity of digital information to exchange, and (iii) the number of computations to perform. Information storage relates to memory space. Information exchanges and computations relate to simulation activity.

Figure 1. Screenshot of a fire spread simulation (visualization tool used: [1])

Focusing on activity makes the simulation systems more efficient. Efficiency of the simulation depends first on the modeling efficiency. The latter necessitates using concise structures reusable simulation components. To be efficient the challenge is to benefit from component advantages while reducing execution overheads induced by the communication between components [2, 3].

Our purpose here is to introduce the activity tracking paradigm as an efficient and reusable top-down specification guide from modeling to simulation. Using this paradigm, models and components designed for simulation should be simpler, clearer and faster.

By the word "paradigm", we consider a set of fundamental critics, rules, analysis, thoughts and structures on which theories and models can be developed. Informal and formal notations, simple algorithms, implementation considerations are provided as "keys" to guide the modeler. At every step of the

modeling process, models are incrementally designed through activity focusing on the relevant structure elements. A guide to reduce complexity from modeling and design points of view is provided. Computation structures of common sense are provided to map more efficiently changes of real systems. This new design perspective aims to be more generic than usual simulation world views and more abstract than reasoning on precise algorithm complexity.

The paper is organized as follows. In section 2, concepts and elements of conventional world views and simulation time flows are abstracted and combined. In section 3, a high-level activity tracking pattern is proposed and activity tracking in a distributed system example is presented. In section 4, activity is described through mathematical structures for both discrete-event and discrete-time driven simulations. In section 5, a special case of distributed systems (spatial systems) is described through cellular models. A mathematical structure for cellular model specifications and implementation design issues are presented. In section 6, an example test bench application is presented and discussed. Finally, conclusion and further works are provided.

CONVENTIONAL COMPOSITE SIMULATION STRATEGIES

A modeling and simulation life cycle is usually composed of three fundamental steps:

1 – Observation and delimitation of the system

For a particular problem, according to *modeling objectives*, by experimenting (questioning) a real (or speculative or imaginative) system, experimental data are collected (or defined) within an experimental frame[1].

2 – System modeling

According to a *first modeling view*, a first model can be defined by abstracting both structure and dynamics of the system in attempting to meet the objectives. This model can be deterministic and/or stochastic, continuous and/or discrete. Discrete models[2] can then be defined hierarchically through many *other modeling views*.

3 – Implementation, verification and validation

Finally, trajectories of a discrete model (which is component-based) are produced by simulation, *i.e.*, the process a model state evolves in time through action mechanisms (the abstract simulators [6]). This permits the verification of the internal structure of the simulation system. A study of these results allows validating, invalidating, learning and improving the model.

Conventional world Views

In the modeling and simulation life cycle, the view concept is fundamental for efficiency and to guide the modeler. A view (or a facet) corresponds to the manner the modeler will answer the problem and account for modeling objectives [5]. Usually, complexity of both structure and behavior of a real complex system cannot be represented faithfully by a computer (as a one-to-one mapping). Different models have to be abstracted, designed and simulated to build many virtual representations, each one improving the understanding of the system. Notice that this multi-model conception can be used broadly as a modeling philosophy in whatever scientific approach. The choice of one view depends on modeler's popular and scientific cultures, his knowledge level and modeling requirements (execution time, precision, etc.) In short: an objective is the question a model is developed to answer; a view is how to build this model (and answer). To reduce complexity for one modeling objective, one view can be further adapted or views can be combined together [7]. The choice of views determines both intelligibility of the model and simulation performance. Finally, adapting J.M. Legay's thinking [8], we can say that, "it is the multiplicity of the point of views [and of the modeling objectives] as well as the confrontation of the results of the corresponding models, which lead to a better knowledge of the system under study." We call such an approach multi-view and multi-objective.

There are three common types of discrete event simulation strategies, also called world views, that are employed in discrete event simulation languages and packages [6, 9, 10]: 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 prescheduling 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 [6].

In this paper, we propose an activity tracking approach that differs from the classical activity scanning strategy in fundamental way to be explained.

[1] An experimental frame describes the conditions under which the system is being observed or experimented. For more information on experimental frames [4, 5].

[2] which can be approximations of the continuous system if this latter does not have analytical solutions.

Simulation Time Flows

In simulation, using digital computers, a distinction is made between the continuous time of reality and simulation time. In digital simulation, the flow of time can only be represented by discrete values that can be obtained by a discretization of the continuous time stream. Simulation time can be managed two ways [9]: by a clock or discrete-events. In a clock (or discrete-time) driven simulation, the simulation time is incremented by a constant 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 represented by integer values (or multiples thereof) and models are called discrete-time models. These models are very prevalent in simulation of physics-based control systems. In a discrete-event driven simulation, the simulation progresses from the occurrence date of an event to that of another, *i.e.*, from one discrete state change to another. In this case, the time base can be represented by real numbers but only a finite set of such values can occur in a finite interval of time. The models are called discrete-event models and have been widely used especially for advanced technological systems such as in manufacturing [11]. They include Timed Petri Nets [12], timed automata [13], and Discrete Event System Specification (DEVS) [6].

The advantage of discrete-event driven simulations is that a simulation model evolves directly from one state change to another. During inactivity periods, no computations are performed. However, at every state change, this necessitates to be able to forecast the occurrence date of the next state change, as well as to deal with event managements in a scheduler. On the other hand, in a discrete-time driven simulation one has to deal with the precision of the time step for the detection of state changes.

The decision to choose a simulation time management depends on the nature of the system and on the modeling objectives. For a system in which every state change occurs at a fixed Δt, discrete-events will produce simulation overhead, and a discrete-time driven simulation will be more efficient. In simple words, we do not have to predict (by computation) what we know to 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 mind and in industrial processes (*e.g.*, robots) or after the discretization of the real and continuous time by humans [14]. 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" [15].

Due to a long history of applications that pre-dates the dawn of the digital computer, discrete-time driven simulations are more pervasive to discrete-event simulations. However, it is increasingly recognized that discrete-events are essential to take into account external events in discrete-time components. Moreover, there is no loss in expressiveness in moving to the discrete-event representation, since every discrete-time state change can be considered as an internal event of a simulation model.

How to choose or to combine modeling views and time flows is not evident. In the next section, these concepts are merged through the activity tracking paradigm.

ACTIVITY TRACKING

According to objectives, tools and views a modeler disposes, a common activity pattern can be used and interpreted to model and simulate distributed interacting components.

Activity tracking pattern

Figure 2 depicts the activity pattern. The latter offers a new perspective to modelers merging the three usual world views (activity- event- and process-oriented strategies). Marks are added to track propagating activity in a component hierarchy. Every usual view is underlined. At every simulation time step, an active set of components is determined.

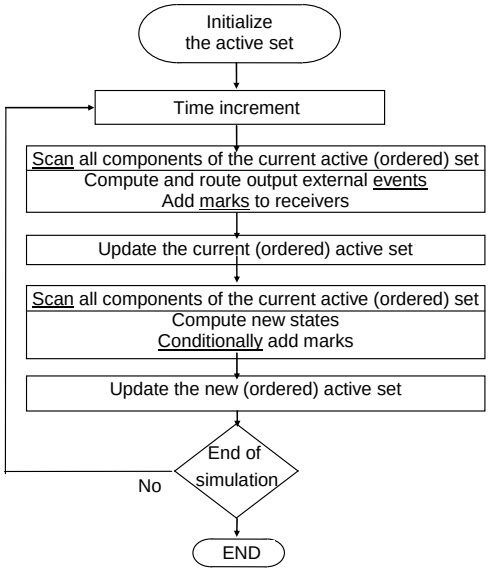

Figure 2. Activity tracking pattern

Using the activity pattern, components can entirely be modeled and simulated through activity tracking (information exchanges and computations), in two steps:

- First, the propagation activity is tracked. Information exchanged between components is routed and computed. The current active set is scanned. Events are routed and output transitions

are computed. Final receivers can be found in the hierarchy using a recursive routing function (for more information, see [2]). The current (ordered) active set is updated including imminent components for external transitions. Order of the active set depends on a tie-breaking function of imminent components (for more information: [6]).

- Second, according to current states and to new information inputs, new states are computed. External and internal transitions (due to external and internal events) of active components are computed. Components changing state significantly are marked to be added to the new (ordered) active set. In a discrete-event driven simulation, the new active set corresponds to a scheduler and active components are marked to execute further their internal transition function (corresponding to an internal event occurrence). In this case, the current active (ordered) set is a sub-set of the scheduler, which corresponds to components active at the current simulation time.

Activity Tracking in Distributed Systems

When distributed components exchange information together, the determination of active components depends on the simulation time management. According to the latter different mechanisms need to be specified to represent the behavior to be simulated. The following simple example is used as a generic case for discussing activity tracking through component modularity and simulation time management. Figure 3 describes the behavior to simulate and then three possible different solutions. The behavior to simulate consists of activating right neighbors of a simple 1-D cellular automata (CA) at different times, reproducing tokens. Crossing times are represented under arrows.

CA [16] is amongst the most well-known paradigms for specifying spatially distributed systems. Standard CA consist of an infinite lattice of discrete identical sites, each site taking on a finite selection of, for instance, integer values. Values of sites evolve synchronously in discrete time steps according to deterministic rules that specify the value of each site in terms of the values of neighboring sites. This basic definition of CA (infinite lattice, neighborhood and rule uniformity of cells, closure of the system to external events, discrete states of cells, etc.) is too limited to specify complicated cellular models. Extensions of basic cellular automata for tracking activity (using a discrete-time or a discrete-event time base) are considered here after.

Using a discrete-time management, as the smallest time precision is *0.1*, the whole discrete-time step has to be: $\Delta t=0.1$. A first simulation of this behavior can be achieved using basic cellular automata. The latter computes every component's state, at every time step, using directly the state of *influencing* left neighboring components. *Local transition functions* consist of:

```
If (influencingComponentState=='getaToken' &&
simulationTimeRequired==true) Then
    myComponentState←newState //get a token
endIf
```

At every time step the whole local transition functions of components are activated, in an inefficient way. Such basic cellular automata do not allow focusing computations only on active components.

Still using a discrete-time base, another solution can be defined to track activity in space. Basic CA use a simple reductionistic view defining exclusively the global behavior as the behavior of the parts (cells). To track activity, a global state transition function is added. An activity state ('*inactive*', '*active*' and '*activeTesting*') is added to the cells. The activity state '*activeTesting*' is used to track new activated components. Then, as a generic *global transition function*, a simple algorithm can be used to determine new active testing components:

```
If (scannedComponentActivityState=='activeTesting' && influencedComponent=='inactive' &&
scannedComponentActivityCondition==true) Then
    scannedComponentActivityState←'active'
    influencedComponentActivityState←'activeTesting'
endIf
```

Here the `scannedComponentActivityCondition` corresponds to the test: `If (simulationTimeRequired==true)`. Once this test is satisfied, the local transition function is computed only for active and testing components. Only components in the activity state 'active testing' are tested. Obviously, another test can be added to turn active components in inactive ones:

```
If (scannedComponentActivityState=='active' &&
scannedComponentActivityCondition==false) Then
    scannedComponentActivityState←'inactive'
endIf
```

Here, this activity end condition will result in an active set of only one active testing component. By the way, external events can change states of components through ports (*cf.* Figure 3, external event on the last right cell).

Using a discrete-event time base, components are autonomous components communicating through ports and external events, scheduling internal events. Cells receiving the required type of external event (token reception) achieve the following algorithm through a *local transition function*:

```
If (inputEventType==true && myActivity
State=='inactive') Then
    //new token
    schedule (internalEvent(delay))
    myActivityState←'active'
    myComponentState←newState //get a token
endIf
```

When the component is reactivated by the internal event reception (after the delay required), the following algorithm is executed *locally*:

```
If (myActivityState=='active') Then
    send (externalEvent)
    myActivityState←'inactive'
endIf
```

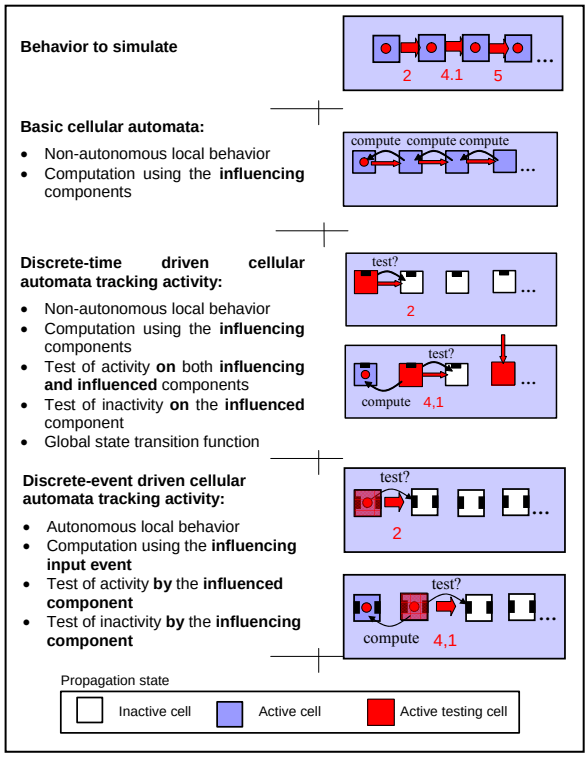

Figure 3. Cellular models

This simple example illustrates the activity detection in distributed systems, in a generic way. According to the simulation time management, autonomous and non-autonomous activity tracking are achieved by atomic components (or cells). In a discrete-event driven simulation, the simulation advances by event scheduling. According to the significance of state changes, components schedule events to be added to the new active set. States of components are encapsulated in discrete-events. Activity is tracked in an autonomous way by local transition functions of both influenced and influencing components. In a discrete-time driven simulation, a global transition function tests both influencing and influenced components.

Through mathematical structures, the next section extends the previous activity tracking principle embedding tests on continuous states. The `"scannedComponentActivityCondition"` corresponds to a test on the significance of state changes

STRUCTURE SPECIFICATION

DEVS and Discrete Time System Specification (DTSS) can be used to describe components activity [6].

Discrete-event Structure Specification

A DEVS atomic component is described by a structure $< X, S, Y, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta >$. X is the set of input events, S is the state set, Y is the set of output events, δ_{int} is the internal transition function, δ_{ext} is the external transition function, δ_{conf} is the confluent transition function, λ is the output function, and ta is the time advance function. Transition functions are triggered by events, and they operate on a bag of inputs (denoted by X^b) and the state of the system when an event occurs.

A DEVS network is defined as $< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$, where, X is the set of input events, Y is the set of output events, D is an index of components, and for each $i \in D$, M_i is a basic DEVS model, I_i is the set of influences of model i. For each $j \in I_i$, Z_{ij} is the i to j translation function.

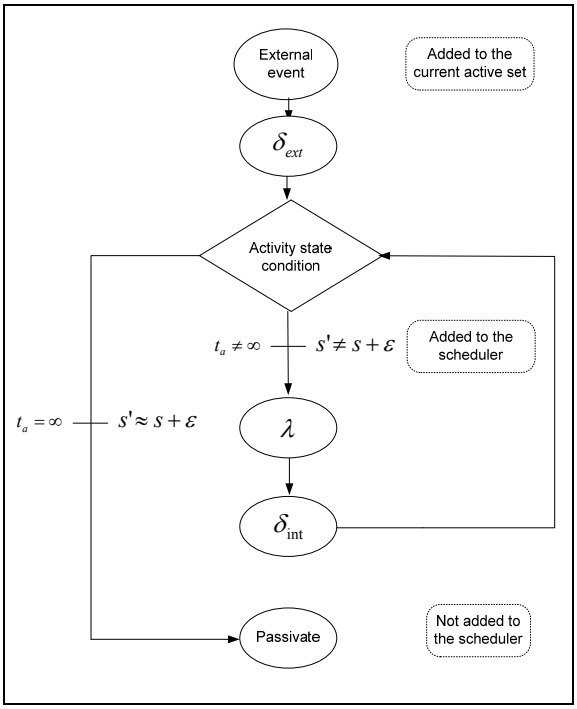

Figure 4. Activity of atomic components in a discrete-event simulation

Using the previous mathematical structure, both component specification and activity detection are illustrated in the flowchart of Figure 4. The latter represents the active behavior of a DEVS atomic component. In a discrete-event driven simulation, first, the atomic component is activated by the occurrence of an external discrete event. Then, the external transition function δ_{ext} calculates the new state s' according to its current state s and to the value of the external

event. If the state changes significantly (*i.e.*, if $s' \neq s + \varepsilon$), the component is added to the scheduler, the time advance function $t_a(s)$ calculates the occurrence of the next internal event. Significance $\varepsilon \in [0, +\infty]$ of state changes is also used in the next discrete-time driven simulation for activity detection. Otherwise (if $s' \approx s + \varepsilon$), the component becomes inactive, the time advance function $t_a(s)$ gives the occurrence time of the next internal event as infinite. The component is not added to the scheduler. When an internal event occurs, the output function λ is executed before the internal transition function δ_{int}. Again, the activity state is tested and occurrence time of the next internal event is then calculated.

Discrete-Time Structure Specification

A DTSS atomic component is described by the structure: $< X, Y, S, \delta_{int}, \lambda, h >$, where (except for sets defined previously) h is the constant time advance.
To detect activity, the network of simple DTSS models is referred to as a Dynamic Structure Discrete Time Network (DSDTN[3]) [17]. Input and output sets are introduced to allow connections with the network. Formally, a DSDTN is a 4-tuple: $<X_{DSDTN}, Y_{DSDTN}, \chi, M_\chi>$, where X_{DSDTN} is the network input values set, Y_{DSDTN} is the network input values set, χ is the name of the DSDTN executive, M_χ is the model of the executive χ. The model of the executive is a modified *DTSS* defined by the 8-tuple: $M_\chi = < X_\chi, S_\chi, Y_\chi, \gamma, \Sigma^*, \delta_{int,\chi}, \lambda_\chi >$, where $\gamma : Q_\chi \rightarrow \Sigma^*$ is the structure function, and Σ^* is the set of network structures. The transition function $\delta_{int,\chi}$ computes the internal executive state s_χ. The network executive structure Σ, at the state $s_\chi \in S_\chi$ is given by $\Sigma = \gamma(s_\chi) = (D, \{M_i\}, \{I_i\}, \{Z_{i,j}\})$, for all $i \in D$, $M_i = < X_i, S_i, Y_i, \delta_{int,i}, \lambda_i, h_i >$, where D is the set of component references, I_i is the set of influencers of model i, and $Z_{i,j}$ is the i to j translation function. Because the network coupling information is located in the state of the executive, transition functions can modify this state and, in consequence, modify the structure of the network. Changes in structure include changes in model interconnections, changes in system definition, and the addition or deletion of system components.

Using the previous mathematical structure, both component specification and activity detection are illustrated in the flowchart of Figure 5. The latter represents the active behavior of a DTSS atomic component. First, the active component is activated. The simulation time is then incremented and the internal transition function δ_{int} is executed. Then, the component's state s is tested through the global transition function $\delta_{int,\chi}$ (whose algorithm is described in the previous section). If the state change is not significant (*i.e.*, if $s' \approx s + \varepsilon$) the component returns in a passive state and is not added to the new active active set. Otherwise, (when $s' \neq s + \varepsilon$), the output function λ of the component is executed. This process (Time increment → Execution of the internal transition function → Activity state condition) is effectuated until the end of the simulation.

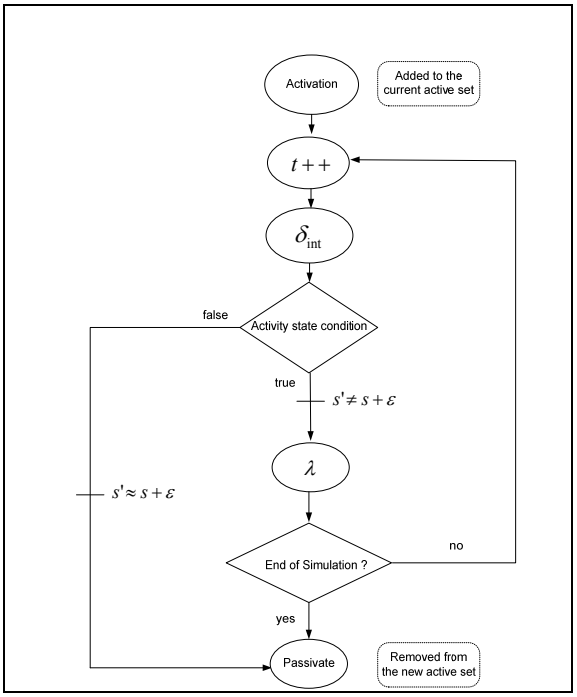

Figure 5. Activity of atomic components in a discrete-time simulation

CELLULAR SYSTEMS

To track activity of cellular systems in a discrete-event driven simulation, usual DEVS models can be used. In a discrete-time driven simulation, we noticed that the implementation of a global transition function at the network level necessitates the use of dynamic structures. We introduce here after a cellular structure specification through dynamic structures. The specification of cells can be reused in a DEVS specification. However, dynamic structure specifications constitute a more powerful formalism than activity specification (as coupling changes). Finally, data structures at the implementation level are presented.

DSCA structure specification

We present here a DSCA specification as a dynamic structure network. A *DSCA* (Dynamic Structure

[3] The dynamic structure formalism is chosen here to allow the specification of the global activity transition function through $\delta_{int,\chi}$.

Cellular Automata) is a structure:

$$DSCA =< X_{DSCA}, Y_{DSCA}, DSCA_\chi >$$

Where, X_{DSCA} and Y_{DSCA}, are respectively output and inputs. Dynamic structure changes are handed by:

$$DSCA_\chi =< X_{DSCA_\chi}, Y_{DSCA_\chi}, S_{DSCA_\chi}, \delta_{DSCA_\chi}, \lambda_{DSCA_\chi}, \tau_{DSCA_\chi} >$$

The *structural state* is defined as $S_{DSCA_\chi} =< D, \{C_i\}, \{I_i\}, \{Z_{i,j}\} >$. For all sub-systems $i \in D$ contains the DSCA references of *active* components, $\{Z_{i,j}\}$ is the set of coupling functions (all cells can be externally connected to both input X_{DSCA} and output Y_{DSCA} of the *DSCA*: $Z_{DSCA \to DSCA_\chi} : X_{DSCA} \to X_{DSCA_\chi}$,

$Z_{DSCA \to c} : X_{DSCA} \to X_c$, $Z_{c \to DSCA} : Y_c \to Y_{DSCA}$ [4], $Z_{DSCA_\chi \to DSCA} : Y_{DSCA_\chi} \to Y_{DSCA}$, and $I_c = \{N_c, DSCA\}$, $I_{DSCA_\chi} = \{\{cell_i\}\}$ where N_c is the neighborhood of a cell *c*. It is a set of pairs representing the relative positions of the neighboring cells *p* and the cell *c*: $N_c = \{(i_p, j_p) / p \in I_c, i_p, j_p \in \mathbb{Z} \wedge i_p, j_p \in [-1,1]\}$.

$\delta_{DSCA_\chi} : X_{DSCA_\chi} \times S_{DSCA_\chi} \to S_{DSCA_\chi}$, is the structural state transition function. According to current structural state and inputs, the transition function can compute new structural states. Changes in structure include changes in cells neighborhoods, changes in cell definitions, and addition or deletion of cells. The structural state transition function is composed of internal and external functions $\delta_{DSCA_\chi} = \{\delta_{\text{int}_{DSCA_\chi}} \cup \delta_{ext_{DSCA_\chi}}\}$. External transitions allow accounting for external events and internal ones for autonomous computations of self states (for more information: [17].)

$\lambda_{DSCA_\chi} : S_{DSCA_\chi} \to Y_{DSCA_\chi}$ is the structural state output function. Through the output function structural states can be sent to other models.

As a minimum assumption, each cell *c* can be specified as an atomic component:

$$C =< X_c, Y_c, S_c, \delta_c, \lambda_c, \tau_c >$$

$S_c =< (m,n), S^{N_c}, phase >$, with

$$\begin{cases} S^{N_c} = \{s_p / p \in I_c\} \\ phase = \{"active", "passive", ...\} \end{cases}$$

Where *(m,n)* are cells' coordinates, and N_c has been defined previously as the neighborhood set.

When receiving or sending its state, a cell is in phase "*active"*, otherwise it is in phase "*passive*".

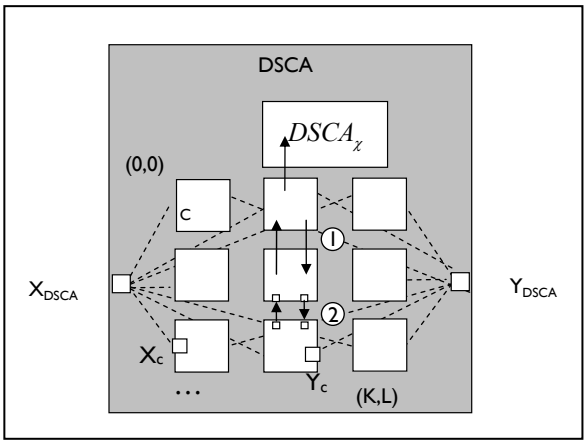

Figure 6. A Dynamic Structure Cellular Automata

$\delta_c : X_c \times S_c \to S_c$ is the transition function[5] composed of internal and external functions $\delta_c = \{\delta_{\text{int}_c} \cup \delta_{ext_c}\}$, where

$\delta_{\text{int}_c} : S_c \to S_c$, and $\delta_{ext_c} : X_c \times S_c \to S_c$.

$\lambda_c : S_c \to Y_c$ is the output function.

$\tau_c : S_c \to \mathbb{R}_0^+$ is a constant time advance.

For a more complex cell, the latter can be decomposed as a network (dynamic structure or not) of sub-components [18]. However, regarding the closure under coupling of DSDEVS, precise network specifications can be expressed by (or is equivalent to) a single atomic specification (more details in [6]).

Efficient Implementation

The relevant issues we retain are: the number of cells (active and passive), the velocity performance and the memory capacity required. In accordance with these issues, two basic approaches can be retained to encapsulate and to manipulate the world data structure [19] (*cf.* Figure 7). The first is spatial oriented and the second is entity oriented. In a spatial oriented approach we can see the world like a grid providing a matrix of positions where a position can be assigned to an entity. This approach is suitable for simulations with a large number of fix entities and where the computer time performance is more important than memory constraints. The second approach is for simulations

[4] For a modular specification, internal couplings of influenced neighboring cells are defined [case (2) of Figure 6] as: $Z_{c \to c'} : Y_c \to X_{c'}$.

[5] Modularity cases: (1) $S_c = S_c^{N_c}$ and $X_c = X_{DSCA}$ (assuming external direct influences of cells), (2) $X_c = X_{DSCA} \times X_c^{N_c}$, with $X_c^{N_c} = \{x_p / p \in I_c\}$ (assuming strong modularity and port interfaces).

with few simulation entities and when we do not want to use the large memory space required by a spatial localization table (matrix of positions). Thus, localization information is saved inside the entity instead of having a matrix of position where a position points to the entity. As a consequence, little memory space will be lost in the simulation implementation. However, the computer time performance will decrease since to get information about environmental position we may have to consult all entities in the worst case.

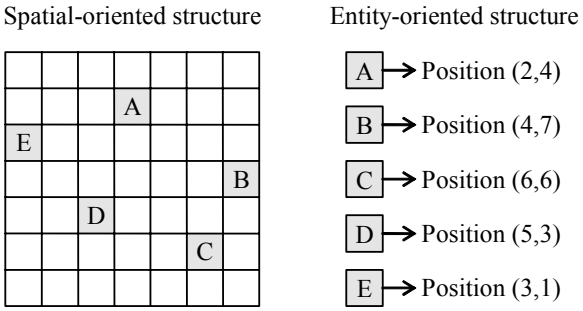

Figure 7. Choices of world data structure for DES

For large cell space simulations, a combination of the two first approaches can be achieved using a spatial-oriented representation for the cellular space and an entity-oriented representation for the active set of cells. Hence, a set of marks of active entities (cells) can be defined using the entity-oriented data structure. In a discrete-time driven simulation, at each time step the global transition function scans the active set adding or removing references of entities according to their state. In a discrete-event driven simulation, cells add and remove their own mark through internal discrete-events.

To reduce execution times, dynamic allocations should be suppressed. Indeed, for significant numbers of object instantiation/deletion, dynamic allocation is inefficient and specialized static allocations have to be designed [20]. A pre-dimensioning via large static arrays and vectors can be easily achieved thanks to current modern computer memory capabilities.

APPLICATIONS DISCUSSION

The sub-section dealing with activity tracking in a distributed system described activity tracking of a discrete variable (presence of a token or not). A deeper and more interesting category is those of continuous systems. Among them, ordinary differential equations constitute a fundamental study case. Using them, physics-based models of the real world can be designed. For activity tracking, continuous systems raise the problem of the detection precision of infinitely rapid state changes. In [21-23], a new method, the quantization, is used to focus precisely computations of discrete-event simulations on the most active segments of a continuous curve. Efficiency advantages and stability of this method are discussed in [24]. Continuous systems can be coupled in partial differential equations (PDE). In [18, 25], PDE have been discretized through cellular models.

Cellular models constitute a representative application case of a distributed system constituted of many interacting sub-systems. For simulation efficiency, cellular systems necessitate focusing computations on active cells among many inactive ones. To achieve this goal, activity tracking has been used in discrete-event driven simulations [3, 26, 27] as well as in discrete-time driven simulations [26, 28, 29]. In both approaches, errors introduced by excluding components considered as inactive from the calculation domain are evaluated. In both approaches parallel and distributed simulations of active components have been achieved [30, 31]. In [30], activity has been used for load balancing of a discrete-event driven simulation. In [31], a fine-grain parallelization of a discrete-time driven simulation has been performed. In every approach focusing on active cells induced a reduction of execution time.

A good example of cellular models obtained from PDE discretization are fire spread simulation [28] (*cf.* Figure 1). After a space discretization, each cell dynamics is modeled by an ordinary differential equation. Inactive cells relate to burned trees (close to the water) and unburned cells to trees away from the fire front. Active cells relate to burning and heated trees.

As discussed previously, computation focus on cells can be achieved in a discrete-time driven simulation (and DSCA) [28] or a discrete-event driven simulation [18]. In both approaches, significance of continuous state changes is used to focus on activity. Increasing temperature gradients lead to reduce the number of transitions and execution times. In the discrete-event approach, more transitions are necessary to focus precisely on activity changes. This is a fine-grain activity tracking. In the discrete-time approach, a coarse-grain activity tracking is used. Cells close to the fire front become active according to the temperature gradient threshold. Then, the temperature curve evolution is calculated according a fix time step. Contrarily to the discrete-event simulation, whatever the activity level, only the first threshold is detected. This is a coarse-grain activity tracking. Figure 8 presents execution times and transition reductions for different quanta in a discrete-event driven simulation. Simulations have been performed on a *1.5GHz Pentium M Centrino*, for *10 000* cells.

Figure 9 presents the linear average relative error (compared with a usual explicit scheme simulation already validated) as the quantum size increases. In fact, increasing the quantum size induces a delay in the fire front propagation.

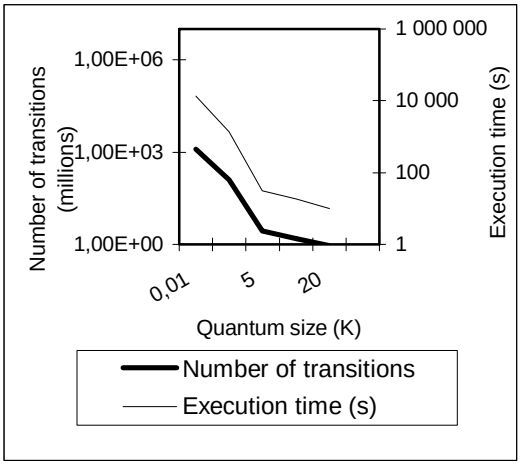

Figure 8. Activity in a discrete-event driven simulation [18]

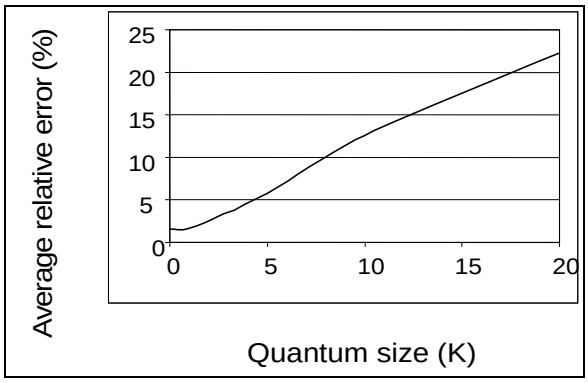

Figure 9. Average relative error in a discrete-event driven simulation [18]

CONCLUSION & FUTURE WORKS

The activity tracking paradigm has been introduced as a new emerging world view for efficient and rigorous modeling and simulation. A major advantage of this paradigm is that component-based simulations are a natural bi-product. Informal, formal and implementation specification levels have been provided to guide modelers in step-by-step model development. Further efforts will be now necessary to elucidate this new paradigm in other applications.

REFERENCES

[1] Muzy, A., D. Hill, M. Joubert, and E. Innocenti. *A post-processed 3D visualization tool for forest fire simulations*. in *IEEE SimuTools*. 2008. Ajaccio. p. Accepted for publication.

[2] 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. p. 273-279.

[3] Hu, X. 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 (ASTC)*. 2004. Arlington, USA. p. 3-8.

[4] Traoré, M. and A. Muzy, *Capturing the dual relationship between simulation models and their context. ScienceDirect TOP25 Hottest Articles.* Simulation Practice and Theory (SIMPRA), 2006. **14**(2): p. 126-142.

[5] Zeigler, B.P., *Multifaceted modelling and discrete event simulation*. Academic press ed. 1984, London.

[6] Zeigler, B.P., H. Praehofer, and T.G. Kim, *Theory of modelling and simulation*. 2000: Academic Press.

[7] Muzy, A., J.d. Lara, and E. Guerra. *Designing PRIMA: A Precise Visual Language for Modeling with Agents, in a Physical environment*. in *MSV'07- The 2007 International Conference on Modeling, Simulation and Visualization Methods, Intel, MIT Media Laboratory…*. 2007. Monte Carlo Resort, Las Vegas, Nevada, USA p. 231-238.

[8] Legay, J.M., *L'expérience et le modèle. Un discours sur la méthode.* INRA ed. 1997, Paris.

[9] Balci, O. *The implementation of four conceptual frameworks for simulation modeling in high-level languages*. in *Winter Simulation Conference*. 1988. p. 287-295.

[10] Coquillard, P. and D. Hill, *Modélisation et Simulation des Ecosystèmes*. Masson ed. 1997.

[11] Ho, Y.-C., *Discrete Event Dynamic Systems: Analyzing Complexity and Performance in the Modern World*, ed. IEEE. 1993.

[12] Wang, J., *Timed Petri Nets: Theory and Application* 1998, Boston: Kluwer Academic Publishers.

[13] Alur, R. and D. Dill, *A theory of timed automata.* Theoretical Computer Science, 1994. **126**: p. 183--235.

[14] Ralston, A. and P. Rabinowitz, *A First Course in Numerical Analysis*. 2001: Dover Publications.

[15] 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.

[16] Wolfram, S., *A new kind of science*. Wolfram Media ed. 2002.

[17] Barros, F.J., *Modelling Formalisms for Dynamic Structure Systems.* ACM Transactions on Modelling and Computer Simulation, 1997. **7**(4): p. 501-515.

[18] Muzy, A., A. Aïello, P.-A. Santoni, B.P. Zeigler, J.J. Nutaro, and R. Jammalamadaka. *Discrete event simulation of large-scale spatial continuous systems*. in *International Conference on Systems, Man and Cybernetics (SMC), IEEE*. 2005. Hawaii, USA. p. 2991-2998.

[19] Campos, A. and D. Hill, *An Agent-based Framework for Visual-Interactive Ecosystem Simulations.* SCS Transactions, 1998. **15**(4): p. 139-152.

[20] Muzy, A., E. Innocenti, D.R.C. Hill, A. Aïello, J.-F. Santucci, and P.A. Santoni, *Object-oriented framework for modelling and simulation of propagation processes: application to a fire spreading.* Environmental modelling and software, 2005. **20**(7): p. 827-842.

[21] Zeigler, B.P., *DEVS theory of quantized systems*. 1998.

[22] Cellier, F. and E. Kofman, *Continuous System Simulation*. 2006: Springer-Verlag New York.

[23] Nutaro, J., *Parallel discrete event simulation with applications to continuous systems*. 2003, University of Arizona: Tucson.

[24] Nutaro, J. and B.P. Zeigler, *On the Stability and Performance of Discrete Event Methods for Simulating Continuous Systems.* Journal of Computational Physics, 2007. **227**(1): p. 797-819.

[25] Jammalamadaka, R., *Activity characterization of spatial models: application to discrete event solution of partial differential equations*. 2003, University of Arizona: Tucson.

[26] Nutaro, J., *A discrete event method for wave simulation.* ACM Trans. Model. Comput. Simul., 2006. **16**(2): p. 174-195.

[27] Nutaro, J., B.P. Zeigler, R. Jammalamadaka, and S. Akrekar. *Discrete event solution of gas dynamics within the DEVS framework: exploiting spatiotemporal heterogeneity*. in *International Conference for Computational Science*. 2003. Melbourne, Australia.

[28] Muzy, A., E. Innocenti, D.R.C. Hill, A. Aïello, J.F. Santucci, and P.A. Santoni. *Dynamic structure cellular automata in a fire spreading application. Chosen as one of the best papers of the conference.* in *First International Conference on Informatics in Control, Automation and Robotics*. 2004. Setubal, Portugal: IEEE/CSS/IFAC/ACM/AAAI/APPIA. p. 143-151.

[29] Jammalamadaka, R. and B.P. Zeigler. *A Generic Pattern for Modifying Traditional PDE Solvers to Exploit Heterogeneity in Asynchronous Behavior*. in *PADS* 2007. p. 45-52.

[30] Park, S. and B.P. Zeigler. *Distributing Simulation Work Based on Component Activity: A New Approach to Partitioning Hierarchical DEVS Models*. in *1st International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*. 2003.

[31] Innocenti, E., A. Muzy, D.R.C. Hill, A. Aïello, and J.F. Santucci. *Design of a Multithreaded Parallel Model for Fire Spread*. in *15 th European Simulation Symposium*. 2003. Delft, The Netherlands: SCS. p. 104-10.