Prof. Dr. François E. Cellier Department of Computer Science ETH Zurich

May 21, 2013

▲日▼▲□▼▲□▼▲□▼ □ ○○○

Discrete Event Simulation

Introduction

Introduction

All of the solvers that we studied until now have the following property in common:

Given the time instant t_{k+1} , the solvers perform a polynomial extrapolation to calculate the values of all state variables at that time instant.

Now we shall study what happens when we reformulate the problem in a reverse fashion. We shall determine *when* a state variable reaches a predetermined value, or more precisely:

Given that the state variable x_i currently assumes the value $x_i(t_k)$, we would like to determine the shortest time distance h, such that $x_i(t_k + h) = x_i(t_k) \pm \Delta Q_i$.

where ΔQ_i is a predetermined *state quantum* associated with the state variable x_i .

As we are simulating a continuous system on a digital computer, we need to discretize something, as no digital computer can compute infinitely many state changes within a finite time interval.

Until now, we always *discretized the time axis*, while *keeping the state variables continuous*. In the sequel, we shall *discretize (quantize) the state variables*, while *keeping the time axis continuous*.

Discrete Event Simulation

Introduction

Introduction II

If we can construct a solver based on this principle, it follows that:

- The integration method will use a variable step size, and the step size in use will depend on the gradient of that state variable.
- The step size h will be different for each state variable x.
- We shall no longer be able to represent the discretized system by a set of difference equations, and we shall lose the linearity of the discretized system when approximating a linear continuous system.

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} \quad
eq \quad \mathbf{x}_{\mathbf{k}+1} = \mathbf{F} \cdot \mathbf{x}_{\mathbf{k}}$$

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● ● ● ● ● ●

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example

Let us start by considering the following first-order system:

$$\dot{x}_{a}(t) = -x_{a}(t) + 10 \cdot \varepsilon(t - 1.76)$$

with initial condition $x_a(t_0 = 0) = 10$.

Rather than simulating this model directly, we shall analyze the following related model:

$$\dot{x}(t) = -\text{floor}[x(t)] + 10 \cdot \varepsilon(t - 1.76)$$

or:

$$\dot{x}(t) = -q(t) + 10 \cdot \varepsilon(t - 1.76)$$

▲□▶ ▲□▶ ▲□▶ ▲□▶ ▲□ ● ● ●

where $q(t) \triangleq \text{floor}[x(t)]$ is the integer part of the variable x(t) > 0.

The latter model can be simulated very easily.

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example II

 $\dot{x}(t) = -q(t) + 10 \cdot \varepsilon(t - 1.76)$; q(t) = floor[x(t)]; x(0) = 10



▲日▼▲□▼▲□▼▲□▼ □ ○○○

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example II

 $\dot{x}(t) = -q(t) + 10 \cdot \varepsilon(t - 1.76)$; q(t) = floor[x(t)]; x(0) = 10



▲日▼▲□▼▲□▼▲□▼ □ ○○○

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example II

 $\dot{x}(t) = -q(t) + 10 \cdot \varepsilon(t - 1.76)$; q(t) = floor[x(t)]; x(0) = 10



◆□▶ ◆□▶ ◆三▶ ◆三▶ 三三 のへで

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example II

 $\dot{x}(t) = -q(t) + 10 \cdot \varepsilon(t - 1.76)$; q(t) = floor[x(t)]; x(0) = 10



Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example III



- We were able to complete the simulation in 17 very simple steps, thereby obtaining the exact solution of the quantized system.
- The solution of the quantized system is similar to that of the original continuous system.

・ロト ・ 一下・ ・ ヨト・

э

Discrete Event Simulation

Space Discretization: A Simple Example

Space Discretization: A Simple Example IV



▲□▶ ▲□▶ ▲□▶ ▲□▶ ▲□ ● ● ●

Discrete Event Simulation

Discrete Event Systems and DEVS

Discrete Event Systems

- Clearly, the quantized system is a discrete system. However, it cannot be represented by a set of difference equations, i.e., it is not a discrete-time system.
- We recognize easily that it may be represented as a discrete-event system.
- The model can be encoded using the DEVS formalism.
- DEVS stands for Discrete EVent System specification. The formalism was first introduced in the 1970s by Bernard Zeigler.
- All systems, the input/output behavior of which can be described by sequences of discrete events, can be represented using the DEVS formalism.

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● ● ● ● ● ●

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS Atomic DEVS Models

A *DEVS model* processes a *sequence of input events* and, in reaction to those events and its own *initial discrete state*, generates a *sequence of output events*.



An *atomic DEVS model* is defined by the structure:

 $M = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$

- X is the set of input values.
- Y is the set of output values.
- S is the set of state values.
- $\delta_{int}(), \delta_{ext}(), \lambda()$, and ta() are functions defining the dynamics of the system.

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● ● ● ● ● ●

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS II

The Behavior of an Atomic DEVS Model



- $\delta_{int}(s)$ is the internal transition function.
- δ_{ext}(s, e, x) is the external transition function.
- ta(s) is the time advance function.

▲日▼▲□▼▲□▼▲□▼ □ ○○○

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS III

The Specification of an Atomic DEVS Model

- Each possible state s (s ∈ S) has an associated time advance calculated by the time advance function ta(s) (ta(s) : S → ℜ₀⁺). The time advance is a non-negative real number, determining how long the system remains in a given state in absence of input events.
- If the state adopts the value s₁ at time t₁, after ta(s₁) units of time (i.e., at time t₁ + ta(s₁)), the system performs an *internal transition*, taking it to a new state s₂. The new state is calculated as s₂ = δ_{int}(s₁). Function δ_{int} (δ_{int} : S → S) is called the *internal transition function*.
- ▶ When the state changes its value from s_1 to s_2 , an *output event* is produced with the value $y_1 = \lambda(s_1)$. Function λ ($\lambda : S \to Y$) is called the *output function*. In this way, the functions t_a , δ_{int} and λ define the *autonomous behavior of a DEVS model*.

A D M 4 目 M 4 日 M 4 1 H 4

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS IV

The Specification of an Atomic DEVS Model

When an input event arrives, the state changes instantaneously. The new state value depends not only on the value of the input event, but also on the previous state value and the elapsed time since the last transition. If the system assumes the state value s_2 at time t_2 , and subsequently, an input event arrives at time $t_2 + e < ta(s_2)$ with value x_1 , the new state is calculated as $s_3 = \delta_{ext}(s_2, e, x_1)$. In this case, we say that the system performs an *external transition*. Function δ_{ext} ($\delta_{ext} : S \times \Re_0^+ \times X \to S$) is called the *external transition function*. No output event is produced during an external transition.

A D M 4 目 M 4 日 M 4 1 H 4

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS V The Specification of an Atomic DEVS Model

Let us consider the following simple example: A system receives positive numbers in an asynchronous way. After it received a number x, it generates an output event with the number x/2 after $3 \cdot x$ time units.

A DEVS model that correctly represents this behavior is the following:

$$M_F = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \text{ where}$$

$$X = Y = S = \Re^+$$

$$\delta_{int}(s) = \infty$$

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

$$\lambda(s) = s/2$$

$$ta(s) = 3 \cdot s$$

Observe that the state can assume a time advance equal to ∞ . When this occurs, we say that the system is in a *passive state*, since it will no longer change its state, unless and until it receives an input event.

Discrete Event Simulation

Discrete Event Systems and DEVS

The Definition of DEVS VI

The Specification of an Atomic DEVS Model

Let us analyze what happens with the model M_1 when it receives an input event trajectory. Consider for instance that input events occur at times t = 1, t = 3, and t = 10 with the values 2, 1, and 5, respectively. Suppose that initially we have t = 0, $s = \infty$ and e = 0.

Then, the following behavior would be observed:

time t = 0: time $t = 3^-$: $s = \infty$ s = 2e = 0e = 2 $ta(s) = ta(\infty) = \infty$ time t = 3: $s = \delta_{ext}(s, e, x) = \delta_{ext}(2, 2, 1) = 1$ time $t = 1^{-1}$ $s = \infty$ e = 1time $t = 3^+$: s = 1time t = 1e = 0 $s = \delta_{ext}(s, e, x) = \delta_{ext}(\infty, 1, 2) = 2$ ta(s) = ta(1) = 3time $t = 1^+$. time t = 6s = 2output event with value $\lambda(s) = \lambda(1) = 0.5$ e = 0 $s = \delta_{int}(s) = \delta_{int}(1) = \infty$ ta(s) = ta(2) = 6

◆□ > ◆□ > ◆臣 > ◆臣 > ─ 臣 ─ のへ⊙

Discrete Event Simulation

Coupled DEVS Models

Coupled DEVS Models

Atomic DEVS models can be coupled to form more complex models. The most simple manner for defining the coupling between DEVS models is through the use of *input and output ports*.



We notice the following couplings:

- from the input port in₀ of model N to the input port in₀ of model M_a,
- from the output port out₁ of model M_a to the input port in₀ of model M_b,
- from the output port out₀ of model M_a to the output port out₀ of model N,

▲ロ ▶ ▲周 ▶ ▲ 国 ▶ ▲ 国 ▶ ● ● ● ● ● ●

etc.

The resulting coupled model N can be used as if it were a new atomic model.

Coupled DEVS Models

Example: DEVS Model of a Static Function

Let us consider a system that calculates a *static function* $f(u_0, u_1)$, where u_0 and u_1 are real-valued piecewise constant trajectories generated by other subsystems. We can represent piecewise constant trajectories by *sequences of events*, if we relate each event to a change in the trajectory value.

Using this idea, we can build the following atomic DEVS model:

$$\begin{split} M_F &= (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \text{ where } \\ X &= Y = \Re \times \mathbb{N}_0 \\ S &= \Re^2 \times \Re^+_0 \\ \delta_{int}(s) &= \delta_{int}(u_0, u_1, \sigma) = (u_0, u_1, \infty) \\ \delta_{ext}(s, e, x) &= \delta_{ext}(u_0, u_1, \sigma, e, x_V, p) = \tilde{s} \\ \lambda(s) &= \lambda(u_0, u_1, \sigma) = (f(u_0, u_1), 0) \\ ta(s) &= ta(u_0, u_1, \sigma) = \sigma \end{split}$$

where:

$$\tilde{s} = \left\{ \begin{array}{ll} (x_v,\,u_1,\,0) & \text{ if } p = 0 \\ (u_0,\,x_v,\,0) & \text{ otherwise} \end{array} \right.$$

Some considerations concerning this model:

- The input and output events carry, beside from the value of the signal itself, an integer number that indicates the corresponding port.
- The discrete state contains three components: u₀, u₁, and σ. The first two maintain the last value received for u₀(t) and u₁(t), whereas σ indicates the time interval until the next output event.
- When an input event arrives, it is assigned the value σ = 0. In this way, an immediate output event is being scheduled.

Simulation of DEVS Models

Simulation of DEVS Models

DEVS models can be simulated with a simple ad-hoc program written in any language. In fact, the simulation of a DEVS model is not much more complicated than that of a discrete-time model.

A basic algorithm that may be used for the simulation of a coupled DEVS model can be described by the following steps:

- 1. Identify the atomic model that, according to its time advance and elapsed time, is the next to perform an internal transition. Call the system d^* , and let t_n be the time of the aforementioned transition.
- 2. Advance the simulation clock t to $t = t_n$ and execute the internal transition function of model d^* .
- 3. Propagate the output event produced by d^* to all atomic models connected to it through its output ports while executing the corresponding external transition functions. Then return to step 1 above.

Simulation of DEVS Models

Simulation of DEVS Models II

One of the simplest ways for implementing these steps is by writing a program with a hierarchical structure equivalent to the hierarchical structure of the model to be simulated.

A routine called *DEVS-simulator* is associated with each *atomic DEVS model*, and a different routine called *DEVS-coordinator* is related to each *coupled DEVS model*. At the top of the hierarchy, there is a routine called *DEVS-root-coordinator* that manages the global simulation time.



There exist several software tools that support directly the simulation of DEVS models. The one that we shall be using is called **PowerDEVS**. It was developed by *Ernesto Kofman* at the Universidad Nacional de Rosario (Argentina). It is the DEVS modeling and simulation environment that is most suitable for our purposes.

Discrete Event Simulation

LDEVS and Continuous System Simulation

DEVS and Continuous System Simulation

In the example of the DEVS model of the static function, we represented piecewise constant trajectories as sequences of events. The same idea can also be used to *approximate continuous systems* using DEVS.

We can divide the quantized continuous system into:

a dynamic system:

 $\dot{x}(t) = d_x(t)$ q(t) = floor[x(t)]

and a *static function*:

$$d_{x}(t) = -q(t) + u(t)$$

where $u(t) = 10 \cdot \varepsilon(t - 1.76)$.

The system can be represented using the following *block diagram*:



Each of the two subsystems has *input and output trajectories that are piecewise constant*. It is thus possible to represent them through DEVS models.

Discrete Event Simulation

DEVS and Continuous System Simulation

DEVS Models of Quantized Systems

The static function can be represented using the DEVS model M_F introduced earlier.

The *dynamic system* can be represented by the following DEVS model:

$$\begin{split} &M_{Ql} = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \text{ where} \\ &X = Y = \Re \times \mathbb{N} \\ &S = \Re^2 \times \mathbb{Z} \times \Re_0^+ \\ &\delta_{int}(s) = \delta_{int}(x, d_x, q, \sigma) = (x + \sigma \cdot d_x, d_x, q + \operatorname{sign}(d_x), \frac{1}{|d_x|} \\ &\delta_{ext}(s, e, x) = \delta_{ext}(x, d_x, q, \sigma, e, x_v, p) = (x + e \cdot d_x, x_v, q, \tilde{\sigma}) \\ &\lambda(s) = \lambda(x, d_x, q, \sigma) = (q + \operatorname{sign}(d_x), 0) \\ &ta(s) = ta(x, d_x, q, \sigma) = \sigma \end{split}$$

where:

$$\tilde{\sigma} = \begin{cases} \begin{array}{cc} \frac{q\!+\!1\!-\!x}{x_V} & \text{if } x_V > 0 \\ \frac{q\!-\!x}{x_V} & \text{if } x_V < 0 \\ \infty & \text{otherwise} \end{array} \end{cases}$$

A D M 4 目 M 4 日 M 4 1 H 4

Discrete Event Simulation

DEVS and Continuous System Simulation

PowerDEVS Model of a Quantized System

The DEVS models M_F (called *static function*) and M_{QI} (called *quantized integrator*) are graphically represented as *PowerDEVS blocks*.

The blocks can then be graphically coupled to each other:



ヘロト 人間ト ヘヨト ヘヨト

э

and the system can be simulated easily.

Discrete Event Simulation

DEVS and Continuous System Simulation

Quantized Systems: Generalization

We can generalize this idea:

Given the *time-invariant continuous system* (state-space model):

$$\dot{x}_{a_1} = f_1(x_{a_1}, x_{a_2}, \cdots, x_{a_n}, u_1, \cdots, u_m)$$

$$\vdots$$

$$\dot{x}_{a_n} = f_n(x_{a_1}, x_{a_2}, \cdots, x_{a_n}, u_1, \cdots, u_m)$$

The system can be approximated by the following *quantized system*:

$$\dot{x}_1 = f_1(q_1, q_2, \cdots, q_n, u_1, \cdots, u_m)$$

$$\vdots$$

$$\dot{x}_n = f_n(q_1, q_2, \cdots, q_n, u_1, \cdots, u_m)$$

which can be represented by the following *block diagram*:



▲日▼▲□▼▲□▼▲□▼ □ ○○○

We can model a generic time-invariant quantized system using DEVS models of the static function and quantized integrator types.

Discrete Event Simulation

LDEVS and Continuous System Simulation

Quantized Systems: Illegitimacy

Unfortunately, there is a problem with the *legitimacy* of the resulting DEVS model.

A DEVS model is said to be illegitimate if it performs an infinite number of transitions in a finite interval of time.

Let us consider the quantized system:

$$\dot{x}(t) = -q(t) + 9.5$$
; $q(t) = floor[x(t)]$

with initial condition x(0) = 10:

- At t = 0, we have q = 10 and thus $\dot{x}(0) = -10 + 9.5 = -0.5$.
- Consequently, at $t = 0^+$, we have x(t) = 9.999... and therefore q(t) = 9.
- This means that $\dot{x}(0) = -9 + 9.5 = +0.5$.
- As a consequence, we get immediately x(t) = 10 and thus return to the initial situation.

We notice that q(t) oscillates between 10 and 9 with *infinite frequency*. For this reason, the DEVS model enters an *infinite loop*, and the simulation cannot advance.

Luckily, this problem can be solved easily by adding hysteresis.

Quantized State Systems

Quantization Functions with Hysteresis

If we add *hysteresis* to the relationship between x(t) and q(t), the oscillations in q(t) can only be produced by *large oscillations* in x(t) that cannot occur instantaneously, as long as the magnitude of the state derivatives remains bounded.

Definition (Function of Quantization with Hysteresis)

Given an ordered sequence of increasing real-valued numbers $(\ldots, Q_{-1}, Q_0, Q_1, \ldots)$, we say that q(t) is related to x(t) through a quantization function with hysteresis, if:

$$q(t) = \begin{cases} Q_m & \text{if } t = t_0 & \wedge & Q_m \leq x(t_0) < Q_{m+1} \\ Q_{k+1} & \text{if } x(t) = Q_{k+1} & \wedge & q(t^-) = Q_k \\ Q_{k-1} & \text{if } x(t) = Q_k - \varepsilon_k & \wedge & q(t^-) = Q_k \\ q(t^-) & \text{otherwise} \end{cases}$$

The discrete values Q_k are called *quantization levels*, and the distance $Q_{k+1} - Q_k$ is called *quantum*. The quantum is often chosen constant. ε_k is the *hysteresis width*.

Discrete Event Simulation

Quantized State Systems

Quantization Functions with Hysteresis II

The graph depicted below shows a *quantization function with hysteresis* with a *uniform quantum*.



Discrete Event Simulation

Quantized State Systems

QSS Method: Definition

Given the *time-invariant continuous system*:

$$\dot{x}_{a_1} = f_1(x_{a_1}, x_{a_2}, \cdots, x_{a_n}, u_1, \cdots, u_m)$$

$$\vdots$$

$$\dot{x}_{a_n} = f_n(x_{a_1}, x_{a_2}, \cdots, x_{a_n}, u_1, \cdots, u_m)$$

approximated by the *quantized state* system (QSS):

$$\dot{x}_1 = f_1(q_1, q_2, \cdots, q_n, u_1, \cdots, u_m)$$

$$\vdots$$

$$\dot{x}_n = f_n(q_1, q_2, \cdots, q_n, u_1, \cdots, u_m)$$

where each q_i is related to x_i by a *hysteretic quantization function*.

The QSS can be represented by the following *block diagram*:



As before, the QSS can be subdivided into *static functions* and *quantized integrators*.

▲日▼▲□▼▲□▼▲□▼ □ ○○○

Discrete Event Simulation

Quantized State Systems

DEVS Representation of a QSS

The DEVS models of the *static functions* are the same as before (M_F) .

The DEVS model of the *quantized integrators* changes a bit due to the presence of *hysteresis*:

$$\begin{split} M_{HQI} &= (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \text{ where} \\ X &= Y = \Re \times \mathbb{N}; \quad S = \Re^2 \times \mathbb{Z} \times \Re_0^+ \\ \delta_{int}(s) &= \delta_{int}(x, d_x, k, \sigma) = (x + \sigma \cdot d_x, d_x, k + \operatorname{sign}(d_x), \sigma_1) \\ \delta_{ext}(s, e, x_u) &= \delta_{ext}(x, d_x, k, \sigma, e, x_v, p) = (x + e \cdot d_x, x_v, k, \sigma_2) \\ \lambda(s) &= \lambda(x, d_x, k, \sigma) = (Q_{k+\operatorname{sign}(d_x)}, 0) \\ ta(s) &= ta(x, d_x, k, \sigma) = \sigma \end{split}$$

with:

$$\sigma_1 = \begin{cases} \begin{array}{c} \frac{Q_{k+2} - (x+\sigma \cdot d_X)}{d_X} & \text{if } d_X > 0 \\ \frac{(x+\sigma \cdot d_X) - (Q_{k-1} - \varepsilon)}{|d_X|} & \text{if } d_X < 0 \\ \end{array} & \begin{array}{c} \sigma_2 = \begin{cases} \begin{array}{c} \frac{Q_{k+1} - (x+e \cdot d_X)}{x_V} & \text{if } x_V > 0 \\ \frac{(x+e \cdot d_X) - (Q_k - \varepsilon)}{|X_V|} & \text{if } x_V < 0 \\ \end{array} \\ \\ \infty & \text{if } x_V = 0 \end{cases}$$

A D M 4 目 M 4 日 M 4 1 H 4

Discrete Event Simulation

Quantized State Systems

Simulation with QSS

- In order to simulate a model using the QSS algorithm, we begin by choosing the quantum to be used by each state variable, i.e., by each hysteretic quantized integrator.
- We then would need to program the static functions and the hysteretic quantized integrators.
- However, PowerDEVS already comes with a library of pre-coded models of hysteretic quantized integrators (the user only needs to choose the quantum) and many different frequently used static functions (summers, limiters, etc.).
- It usually suffices to graphically construct the *block diagram* describing the system, choosing the quantum used by each of the state variables, and dragging the appropriate static functions from the graphical library and dropping them into the diagram window.
- It should be mentioned, however, that the QSS algorithm is independent of DEVS. We chose DEVS for the implementation of the QSS method, because DEVS simplified our work. However, we could have programmed the QSS method also independently of DEVS using any other event description formalism.

Discrete Event Simulation

Quantized State Systems

Simulation with QSS: An Illustrative Example

Let us consider the following second-order system and its QSS approximation:

$$egin{array}{rcl} \dot{x}_{a_1}(t) &=& x_{a_2}(t) & \dot{x}_1(t) &=& q_2(t) \ \dot{x}_{a_2}(t) &=& 1-x_{a_1}(t)-x_{a_2}(t) & \dot{x}_2(t) &=& 1-q_1(t)-q_2(t) \end{array}$$

To simulate this system, we simply construct the *block diagram* using the hysteretic quantized integrator and the appropriate static functions of **PowerDEVS**:



- The *initial conditions* are parameters of the integrators (in our case: x₁(0) = x₂(0) = 0).
- ► The *quantum* and the *hysteresis* are parameters of each integrator (here: $Q_{k+1} Q_k = \Delta Q = \epsilon_k = 0.05$).
- The QSS method intrinsically exploits *sparsity* (events are only propagated between directly connected blocks).

Discrete Event Simulation

Quantized State Systems

Simulation with QSS: An Illustrative Example II

The *simulation results* are shown below:



- The trajectories of the state variables x_i(t) are piecewise linear.
- The trajectories of the quantized states q_i(t) are piecewise constant.
- The presence of the hysteresis is easy to observe where the signs of the state derivatives x_i(t) change.
- The obtained solution is quite close to the analytical solution.

Discrete Event Simulation

Conclusions

Conclusions

- In this presentation, we introduced a new type of discretization. Instead of discretizing the time, we proposed a quantization of the state variables.
- We then outlined a new numerical integration algorithm based on this idea, the QSS algorithm, that operates on quantized states with hysteresis.
- QSS simulations are *intrinsically asynchronous*. Each state variable changes its value independently of the other state variables.
- The QSS algorithm exploits the *sparsity of the model topology*. Events are propagated only between blocks that are directly connected.
- Unfortunately, the QSS algorithm cannot be easily programmed as a Matlab function. Instead, we also introduced a new tool, PowerDEVS, that has been specifically designed for the numerical simulation of continuous systems using QSS algorithms.

A D M 4 目 M 4 日 M 4 1 H 4

Discrete Event Simulation

References

References

- Cellier, F.E., E. Kofman, G. Migoni, and M. Bortolotto (2008), "Quantized State System Simulation," Proc. GCMS'08, Grand Challenges in Modeling and Simulation, part of SCSC'08, Summer Computer Simulation Conference, Edinburgh, Scotland, pp. 504-510.
- Wang, Q., and F.E. Cellier (1990), "Time Windows: An Approach to Automated Abstraction of Continuous-time Models into Discrete-event Models," Intl. J. General Systems, 19(3), pp. 241-262.
- Wang, Qingsu (1989), Management of Continuous System Models in DEVS-Scheme: Time Windows for Event-Based Control, MS Thesis, Dept. of Electr. & Comp. Engr., University of Arizona, Tucson, AZ.

A D M 4 目 M 4 日 M 4 1 H 4