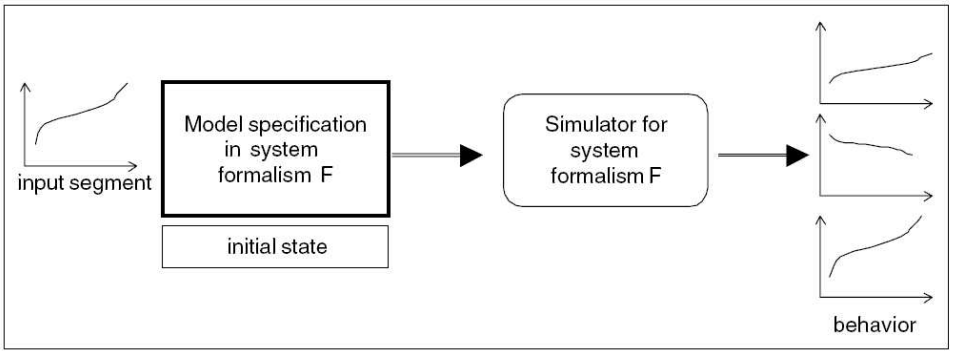
Slide 0

Chap.8 Simulators for Basic Formalisms

Introduction

- simulation
  - is a transformation from higher-level system structure knowledge to the I/O relation observation level
  - is to generate the corresponding state and output trajectories

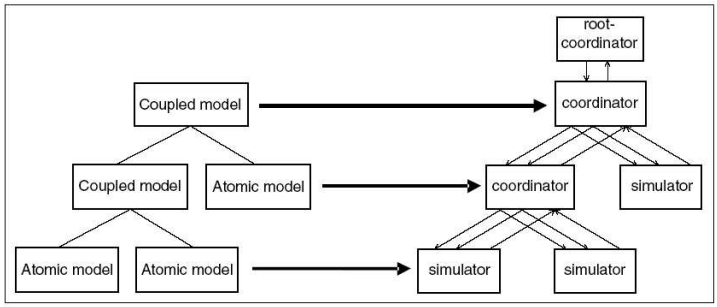
Slide 1



Introduction (Cont'd)

Slide 2

- system formalism
  - basic specification is supported by a class of atomic models
  - atomic class has its own *simulator* class
  - coupled model structure is supported by a class of coupled models
  - coupled model class has its own simulator class, called *coordinator*
  - mapping of a hierarchical models to an abstract simulator

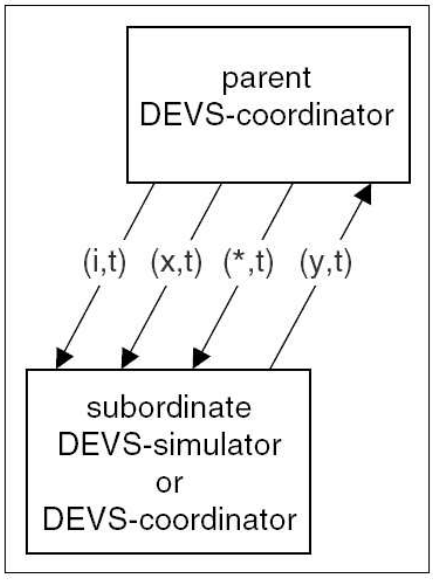


  - at the top, a *root-coordinator* is in charge to initiate the simulation cycles

Simulators for DEVS

Slide 3

- four types of messages



Simulators for DEVS (Cont'd)

Slide 4

- (i, t)
  - an initialization message
  - sent at the initialization time from the parent simulator to all its subordinates
- $(*, t)$
  - scheduling of events is done by the internal state transition message $(*, t)$
  - sent from the coordinator to its imminent child
- (y, t)
  - an output message
  - sent from the subordinates to their parents to notify them of output events
- (x, t)
  - an input message
  - sent from the coordinator to its subordinates to cause external events

Simulator for basic DEVS

Slide 5

- *devs-simulator*
  - interpretation of the dynamics of a DEVS
  - employ two time variables tl and tn
    - tl: hold the simulation time when the last event occurred
    - tn: hold the scheduled time for the next event
    - ☛ $tn = tl + ta(s)$
    - t: global current simulation time
    - e: elapsed time since the last event, $e = t - tl$
    - σ: the time left to the next event, $\sigma = tn - t = ta(s) - e$
    - $\surd$ the tn is sent to its simulator's parent for correct synchronization of events
  - simulator for basic DEVS

Simulator for basic DEVS (Cont'd)

Slide 6

```
Devs-simulator
  variables:
    parent    // parent coordinator
    tl        // time of last event
    tn        // time of next event
    DEVS       // associated model
        with total state (s, e)
    y          // current output value of the associated model
  when receive i-message (i, t) at time t
    tl = t - e
    tn = tl + ta(s)
  when receive *-message (*, t) at time t
    if    t != tn then
      error: bad synchronization
    y = λ (s)
    send y-message (y, t) to parent coordinator
    s = δ_int (s)
    tl = t
    tn = tl + ta(s)
  when receive x-message (x, t) at time t with input value x
    if not (tl ≤ t ≤ tn) then
      error: bad synchronization
    e = t - tl
    s = δ_ext (s, e, x)
    tl = t
    tn = tl + ta(s)
end Devs-Simulator
```

Algorithm 1 Simulator for Basic DEVS.

Simulator for basic DEVS (Cont'd)

Slide 7

* (i, t)
  - when a devs-simulator receives an initialization message
  - it initializes its time of last event $tl = t - e$, t provided by the message
  - the next event time tn is computed by $tn = tl + ta(s)$
  - this tn is sent to the parent coordinator to schedule
* $(*, t)$
  - causes the execution of an internal event
  - when a *-message is received by a devs-simulator
  - it computes the output and carries out the internal transition function
  - the output is sent to the parent coordinator in an output message (y, t)
  - the last event time tl is set to the current time t
  - the next event time tn is computed by $tn = tl + ta(s)$
* (x, t)
  - inform the simulator of an arrival of an input event, x
  - causes the simulator to execute the external transition function
  - the last event time tl is set to the current time t
  - the next event time tn is computed by $tn = tl + ta(s)$

Simulators for modular DEVS networks

Slide 8

- a coordinator
  - is responsible for
    - the correct synchronization of the component simulators
    - handling of external events arriving at the network inputs
  - implements an event list algorithm for the correct synchronization of its components
  - an event list
    - a list of pairs of component simulators with their times of next event
    - sorted by the event times tn, when the tn times are equal, by the *select* function
    - the first component defines the next internal event time, $tn = \min\{tn_d | d \in D\}$
    - the tn is provided to the parent of the coordinator as its next event time
    - the time of last event of the coordinator is $tl = \max\{tl_d | d \in D\}$
  - coordinator for DEVS coupled model

Simulators for modular DEVS networks (Cont'd)

Slide 9

```
Devs-coordinator
  variables:
    DEVN = (X, Y, D, {M_d}, {I_d}, {Z_i,d}, Select)
      //the associated network
    parent        // parent coordinator
    tl            // time of last event
    tn            // time of next event
    event-list
      // list of elements (d, tn_d) sorted by tn_d and Select
    d*            // selected imminent child
when receive i-message (i, t) at time t
  for-each d in D do
    send i-message (i, t) to child d
  sort event-list according to tn_d and Select
  tl = max {tl_d | d ∈ D}
  tn = min {tn_d | d ∈ D}
when receive *-message (*, t) at time t
  if t ! = tn then
    error: bad synchronization
  d* = first (event-list)
  send *-message (*, t) to d*
  sort event-list according to tn_d and Select
  tl = t
  tn = min {tn_d | d ∈ D}
```

Simulators for modular DEVS networks (Cont'd)

Slide 10

```
when receive x-message (x, t) at time t with external input x
  if not (tl ≤ t ≤ tn) then
    error: bad synchronization
  //consult external input coupling to get children influenced by
  //the input
  receivers = {r | r ∈ D, N ∈ I_r, Z_{N,r}(x) ≠ Φ}
  for-each r in receivers
    send x-messages (x_r ,t) with input value x_r = Z_{N,r}(x) to r
  sort event-list according to tn_d and Select
  tl = t
  tn = min {tn_d | d ∈ D}
when receive y-message (y_{d*}, t) with output y_{d*} from d*
  //check external coupling to see if there is an external out-
  //put event
  if d* ∈ I_N & Z_{d*,N}(y_{d*}) ≠ Φ then
    send y-message (y_N ,t) with value y_N = Z_{d*,N}(y_{d*}) to parent
  //check internal coupling to get children influenced by out-
  put y_{d*} of d*
  receivers = {r | r ∈ D, d* ∈ I_r, Z_{d*,r}(y_{d*}) ≠ Φ}
  for-each r in receivers
    send x-messages (x_r,t) with input value x_r = Z_{d*,r}(y_{d*}) to r
end Devs-coordinator
```

Algorithm 2 Coordinator for DEVS coupled model.

Simulators for modular DEVS networks (Cont'd)

Slide 11

* (i, t)
  - when a devs-coordinator receives an i-message
  - it forwards it to each of its children
  - when all the children have handled i-message,

  $\rightarrow$ it sets its time of last event tl to the maximum time of last events

  $\rightarrow$ it sets its time of next event tn to the minimum time of next events
* $(*, t)$
  - when a devs-coordinator receives an $*$-message
  - it transmitted to its imminent child, which is the first in the event list
  - the imminent child executes its state transition and may send output messages (y, t) back
  - after completion of the internal event and all the external events caused by the output of the imminent component, next event time tn is computed

Simulators for modular DEVS networks (Cont'd)

Slide 12

* (y_{d^*}, t)
  - when a devs-coordinator receives an output message (y_{d^*}, t)
  - it decides if it should be transmitted to its parent coordinator, and obtain the children, and their respective input ports with EOC, IC
  - in the first case, an output message (y_N, t) is sent to the parent coordinator
  - in the second case, the output message (y_{d^*}, t) is converted into input message (x_r, t) with $x_r = Z_{d^*,r}(y_{d^*})$, indicating the arrival of external events
* (x, t)
  - when a devs-coordinator receives an input message (x, t) from its parent coordinator
  - it generates an appropriate input message with EIC and sends input messages $(Z_{N,r}(x), t)$ to all children, r, influenced
  - event times are updated

The root-coordinator

Slide 13

- root-coordinator
  - implements the overall simulation loop

```
devs-root-coordinator
  variables:
    t        // current simulation time
    child    // direct subordinate devs-simulator or -coordinator
  t = t0
  send initialization message (i, t) to child
  t = tn of its child
  loop
       send (*, t) message to child
       t = tn of its child
  until end of simulation
end devs-root-coordinator
```

Algorithm 3 Root-coordinator for DEVS abstract simulator.

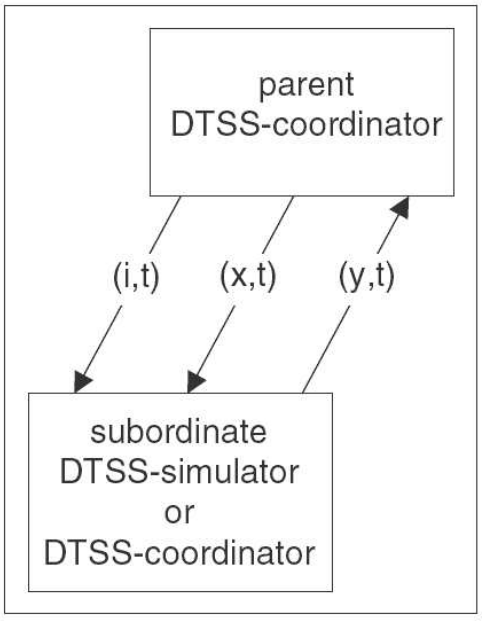
Simulators for DTSS

Slide 14

- simulator
  - at every constant time, the simulator computes
    - the next state by a next state function using the state and the input at the current time
    - and the output based on the current state at any time
  - consists of *dtss-simulators* and *dtss-coordinators* with three types of messages
    - (i, t): sent from the parent simulator to all its subordinates
    - $(*, t)$: sent from the parent to its subordinates to compute their outputs
    - (x, t): sent to its subordinates to trigger the execution of the state transition

Simulators for DTSS (Cont'd)

Slide 15



Simulators for DTSS (Cont'd)

Slide 16

- simulators for atomic DTSS
  - the *dtss-simulator*
    - implement the simulation algorithm for an atomic DTSS (of the Moore-type)
    - provide two messages at each time step
    - $*$-message is used to generate the output but there is no internal transition
    - x-message is used to execute the state transition
    - ☛ DTSS can be viewed as an externally stimulated DEVS with no internal autonomy

Simulators for DTSS (Cont'd)

Slide 17

```
Dtss-moore-simulator
  variables:
    DTSS = (X, Y, Q, δ, λ, h)    // associated model
      with state q
    y     // the current output value of the associated model
    tn    // time of next state transition
when receive message (i, t) at time t
    tn = t
when receive message (*, t) at time t
    if (t != tn) then
      synchronization error
    y = λ (q)
    send y in message (y, tn) to parent
when receive message (x, t) with input value x
    if (t != tn) then
      synchronization error
    q = δ (q, x)
    tn = tn + h
end Dtss-moore-simulator
```

Algorithm 6 Simulator for Moore-type DTSS.

Simulators for DTSS (Cont'd)

- simulators for instantaneous functions
  - function specified system (FNSS)
    - elementary systems that do not own a state
    - compute the output values based on the current input values
    - occur as components in networks of DTSS
    - *fnss-simulator*

Slide 18

```
Fnss-simulator    // for memoryless models
  variables:
    Fnss = (X, Y, λ)    // associated memoryless model
    y    // the current output value of the associated model
  when receive message (x, t) with input value x
    y = λ(x)
    send y in y-message (y, t) to parent coordinator
end Fnss-simulator
```

Algorithm 7 Simulator for memoryless FNSS.

Simulators for DTSS (Cont'd)

- simulators for coupled DTSS
  - define a *coordinator* for networks of DTSS
  - coordinator
    - send $i-$,$*-$, and $x-$messages and receive $y-$messages from them
    - serve as a subordinate simulator
  - dtss-coordinator
    - main task is to schedule the components in the right sequence
    - $*-$message have to compute the output of the network by forwarding the $*-$message to all the Moore-type components
    - the outputs are received in the $y-$messages and are stored (in variables xy_d)

Slide 19

Simulators for DTSS (Cont'd)

Slide 20

```
Dtss-coordinator
  variables:
    N = (X_N, Y_N, D, {M_d}, {I_d}, {Z_d}, h)
        // the network associated with the coordinator
    MOORE = {d ∈ D | d is of type Moore}
    FNSS = {d ∈ D | d is of type FNSS}
    {xy_d | d ∈ D ∪ {N}}
      // variables to store the output values for components d
      // and the network input
    tn    // time of next state transition
  when receive message (i, t) at time t
    for d ∈ D do
      send i-message (i, t) with time t to d
    tn = t
  when receive message (*, t) with time t
    if t != tn
      synchronization error
    for d ∈ MOORE do
      send message (*, t) with time t to d
    while (∃ d ∈ FNSS : ∀i ∈ I_d : xy_i at time t is available
        & d not scheduled for time t) do
      send x-message (x_d, t) with x_d = (Z_d(. . ., xy_i, . . .),
      i ∈ I_d, to d
      mark d scheduled for time t
    endwhile
```

Simulators for DTSS (Cont'd)

Slide 21

```
  when receive x-message (x_N, t) with input value x_N
    if t != tn
      synchronization error
    set xy_N to x_N
    while (∃ d ∈ D : ∀i ∈ I_d : xy_i at time t is available
        & d not scheduled for time t) do
      send x-message (x_d, t) with x_d = (Z_d (. . ., xy_i, . . .),
      i ∈ I_d, to d
      mark d scheduled for time t
  endwhile
  when receive y-message (y_d, t) with output value y_d from com-
  ponent d
    set xy_d to y_d
    if (∀i ∈ I_N : xy_i at time t is available
        & network output not sent for time t) then
      send y-message (y_N, t) with y_N = (Z_N (. . ., xy_i, . . .),
      i ∈ I_N, to parent
      mark network output sent for time t
end Dtss-coordinator
```

Algorithm 9 Coordinator for coupled DTSS.

Simulators for DTSS (Cont'd)

Slide 22

- the root-coordinator

```
dtss-root-coordinator
  t        // current simulation time
  child    // direct subordinate simulator or coordinator

  t = t0
  send initialization message (i, t) to child
  t = tn of its child
  loop
     send (*, t) message to child
     send (x, t) message to child
     t = tn of its child
  until end of simulation

end dtss-root-coordinator
```

Algorithm 10 Root-coordinator for DTSS abstract simulator.

Simulators for DESS

Slide 23

- causal simulator for DESS
  - save the m past values of the model state values, q, derivatives, r, and inputs, x
  - simulation step size h, $t_{i+1} = t_i + h$

$$q(t_{i+1}) = CasualMethod(\quad [q(t_{i-m}), \ldots, q(t_i)], [r(t_{i-m}), \ldots, r(t_i)], \\ [x(t_{i-m}), \ldots, x(t_i)], h)$$

$$y(t_i) = \lambda(q(t_i))$$

Simulators for DESS (Cont'd)

Slide 24

```
Dess-causal-simulator
  variables:
    DESS = (X, Y, Q, f, λ)    // associated model
    [q(t_{i-m}), . . ., q(t_i)]    // is a vector of state values
    [r(t_{i-m}), . . ., r(t_i)]    // is a vector of derivatives
    values
    [x(t_{i-m}), . . ., x(t_i)]    // is a vector of inputs
    h                         // integration stepsize
  when receive message (i, t_i) at time t_i
    initialize [q(t_{i-m}), . . ., q(t_i)],
               [r(t_{i-m}), . . ., r(t_i)],
               [x(t_{i-m}), . . ., x(t_i)]
               using initialization specified by CausalMethod.
  when receive message (*, t_i) at time t_i
    y = λ (q(t_i))
    send y in message (y, t_i) to parent
  when receive message (x, t_i) with input value x
    x(t_i) = x
    q(t_{i+1}) = q(t_i+h) = Causal Method ( [q(t_{i-m}), . . ., q(t_i)],
                                            [r(t_{i-m}), . . ., r(t_i)],
                                            [x(t_{i-m}), . . ., x(t_i)], h)
end Dess-causal-simulator
```

Algorithm 11 Causal simulator for basic DESS.

Simulators for DESS (Cont'd)

Slide 25

- noncausal simulator for DESS
  - predictor phases

$$q^{'}(k+1) = kth - Predictor(\quad [q(t_{i-m}),\ldots,q(t_i),q^{'}(1),\ldots,q^{'}(k)], \\ [r(t_{i-m}),\ldots,r(t_i),r^{'}(1),\ldots,r^{'}(k)], \\ [x(t_{i-m}),\ldots,x(t_i),x^{'}(1),\ldots,x^{'}(k)], \\ h)$$

  - corrector phases

$$q(t_{i+1}) = Corrector(\quad [q(t_{i-m}),\ldots,q(t_i),q^{'}(1),\ldots,q^{'}(n)], \\ [r(t_{i-m}),\ldots,r(t_i),r^{'}(1),\ldots,r^{'}(n)], \\ [x(t_{i-m}),\ldots,x(t_i),x^{'}(1),\ldots,x^{'}(n)], \\ h)$$

    * at the first $*-$message, k is 0 and the output is computed based on the $q(t_i)$
    * at a later with a nonzero k, we have to compute the output for the k-th state estimate $q^{'}(k)$

Simulators for DESS (Cont'd)

Slide 26

```
Dess-non-causal-simulator
  variables:
    DESS = (X, Y, Q, f, λ)     // associated model
    [q(t_{i-m}), . . ., q(t_i)]   // is a vector of past states
                                  // values
    [r(t_{i-m}), . . ., r(t_i)]   // is a vector of past derivatives
                                  // values
    [x(t_{i-m}), . . ., x(t_i)]   // is a vector of past inputs
                                  // values
    [q'(1), . . ., q'(n)]         // is a vector of predicted states
                                  // values
    [r'(1), . . ., r'(n)]         // is a vector of predicted
                                  // derivatives values
    [x'(1), . . ., x'(n)]      // is a vector of predicted inputs
    h                          // integration stepsize
    k                          // indicator for the phase of the
                               // integration

  when receive message (i, t_i) at time t_i
    initialize [q(t_{i-m}),. . ., q(t_i)],
               [r(t_{i-m}), . . ., r(t_i)],
               [x(t_{i-m}), . . ., x(t_i)]
               using initialization specified by integration
               method.

    k = 0
```

Simulators for DESS (Cont'd)

Slide 27

```
  when receive message (*, t_i)
    if k = 0 then y = λ (q(t_i))
             else y = λ (q' (k))
    send y in message (y, t_i) to parent

  when receive message (x, t_i)
    if k = 0 then x(t_i) = x
             else x' (k) = x
    k = (k + 1) mod (n+1)
    if k > 0 then
      q'(k) = kth-Predictor
      ( [q(t_{i-m}), . . ., q(t_i), q'(1), . . ., q'(k-1)],
      [r(t_{i-m}), . . ., r(t_i), r'(1), . . ., r'(k-1)],
      [x(t_{i-m}), . . ., x(t_i), x'(1), . . ., x'(k-1)],
      h )

    else
    q(t_{i+1}) = Corrector
                    ( [q(t_{i-m}), . . ., q(t_i), q'(1), . . ., q'(n)],
                      [r(t_{i-m}), . . ., r(t_i), r'(1), . . ., r'(n)],
                      [x(t_{i-m}), . . ., x(t_i), x'(1), . . ., x'(n)],
                      h )

    t_i = t_i + h

end Dess-non-causal-simulator
```

Algorithm 12 Noncausal simulator for basic DESS.