Slide 0

Chap.4 Introduction to Discrete Event System Specification (DEVS)

Classic DEVS system specification

Slide 1

- a *discrete event system specification (DEVS)* is a structure

  $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$

  where

  X is the set of input values

  S is a set of states

  Y is the set of output values

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

  $\delta_{ext} : Q \times X \to S$ is the *external transition* function, where

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

  e is the *time elapsed* since last transition

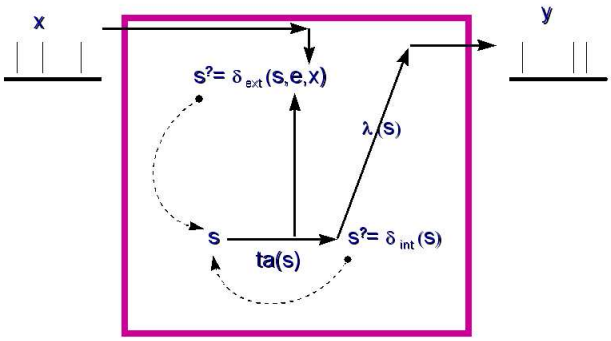
  $\lambda : S \to Y$ is the output function

  $ta : S \to R^{+}_{0,\infty}$ is the set positive reals with 0 and ∞

Classic DEVS system specification (Cont'd)

Slide 2

- interpretation of DEVS



Classic DEVS system specification (Cont'd)

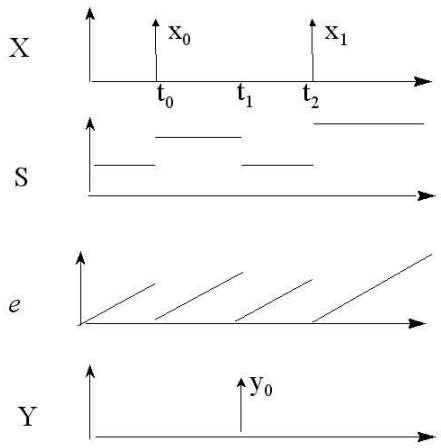
Slide 3

- if no external event occurs, the system will stay in state s for time $ta(s)$
  - if $ta(s) = 0 \rightarrow$so short that no external events can intervene
    s is *transitory* state
  - if $ta(s) = \infty \rightarrow$stay in s forever unless an external event interrupts
    s is *passive* state
  - when the elapsed time e is expired, $e = ta(s)$
    - the system outputs the value, $\lambda(s)$, and changes to state $\delta_{int}(s)$
    - output is only possible just before internal transitions
- if an external event $x \in X$ occurs before the expiration time (total state (s, e))
  - the system changes to state $\delta_{ext}(s, e, x)$

Classic DEVS system specification (Cont'd)

Slide 4

- behavior of a DEVS



DEVS examples

Slide 5

- passive DEVS
  - does not respond with outputs no matter what the input trajectory is

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

    where

    $X = R$

    $Y = R$

    $S = \{\text{"passive"}\}$

    $\delta_{ext}(\text{"passive"}, e, x) = \text{"passive"}$

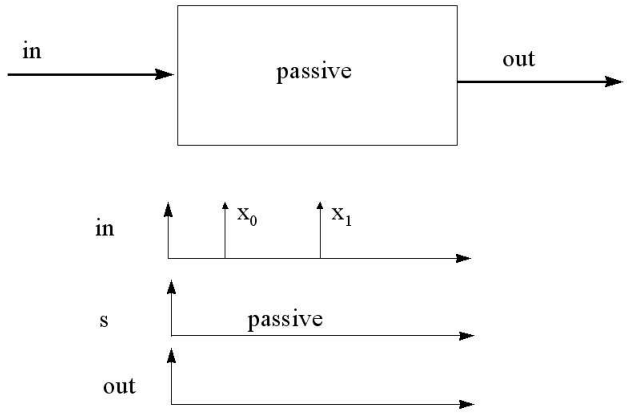
    $\delta_{int}(\text{"passive"}) = \text{"passive"}$

    $\lambda(\text{"passive"}) = \varnothing$

    $ta(\text{"passive"}) = \infty$

DEVS examples (Cont'd)

Slide 6



DEVS examples (Cont'd)

Slide 7

- storage DEVS
  - respond to its input and store it forever or until the next input comes
  - need second input for querying
  - because there is only one input, the input value of zero means this query

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

where

$X = R$

$Y = R$

$S = \{\text{"passive","respond"}\} \times R_0^+ \times R - \{0\}$

$$\delta_{ext}(\text{"passive"}, \sigma, \text{store}, e, x) = \begin{cases} (\text{"passive"}, \sigma - e, x) & \text{if } x \neq 0 \\ (\text{"respond"}, \text{respond\_time}, \text{store}) & \text{if } x = 0 \end{cases}$$

$\delta_{ext}(\text{"respond"}, \sigma, \text{store}, e, x) = (\text{"respond"}, \sigma - e, \text{store})$

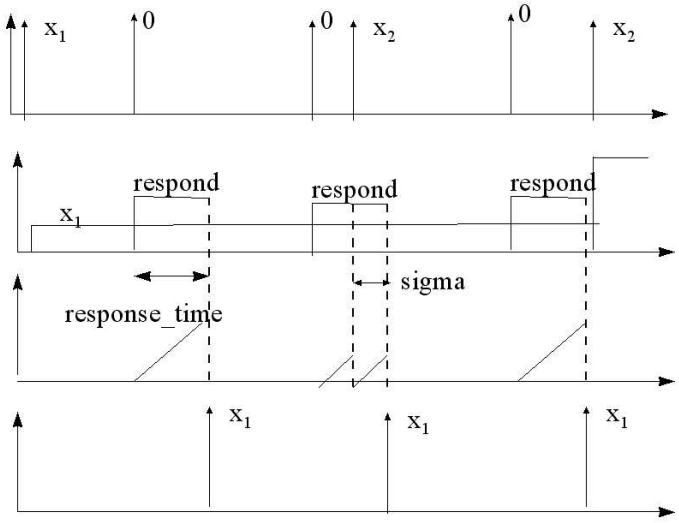
$\delta_{int}(\text{"respond"}, \sigma, \text{store}) = (\text{"passive"}, \infty, \text{store})$

$\lambda(\text{"respond"}, \sigma, \text{store}) = \text{store}$

$ta(\text{phase}, \sigma, \text{store}) = \sigma$

DEVS examples (Cont'd)

Slide 8



DEVS examples (Cont'd)

Slide 9

- three state variables
  - *phase* with values "passive","respond"
  - σ having positive real values is the time remaining in the current state
  - *store* having real values other than zero
- four cases
  - when a zero input arrives at "passive" phase
    - σ is set to *response_time* and the "respond" phase is entered
  - when a nonzero input arrives at "respond" phase
    - the input is ignored
  - when the *response_time* has elapsed
    - the output function produces the stored value
    - the internal transition dictates a return to the passive state
  - when a nonzero input arrives just as the response period has elapsed
    - classic DEVS and parallel DEVS differ

DEVS examples (Cont'd)

Slide 10

- generator
  - generate outputs with a specific periods
  - have two basic state variables: $phase$ and σ

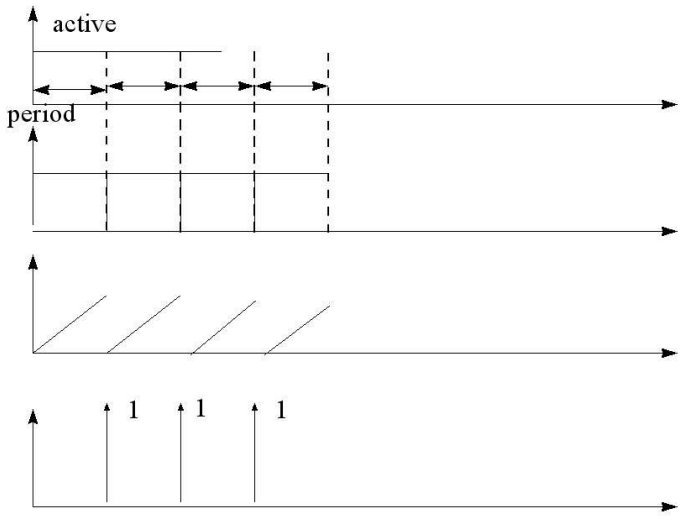
    $DEVS = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$

    where

    $X = \{\}$
    $Y = \{1\}$
    $S = \{\text{"passive"}, \text{"active"}\} \times R^+$
    $\delta_{int}(\text{phase}, \sigma) = (\text{"active"}, period)$
    $\lambda(\text{"active"}, \sigma) = 1$
    $ta(\text{phase}, \sigma) = \sigma$

DEVS examples (Cont'd)

Slide 11



DEVS examples (Cont'd)

- binary counter
  - output a "one" for every two "one"s that it receives
  - maintain a count (modulo 2) of the "one"s it has received to date
  - when it receives a "one" that makes its count even, it goes into "active" to generate the output

Slide 12

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

where

$X = \{1\}$

$Y = \{1\}$

$S = \{\text{"passive"}, \text{"active"}\} \times R_0^+ \times \{1\}$

$\delta_{ext}(\text{"passive"}, \sigma, \text{count}, e, x) =$

$$\begin{cases} (\text{"passive"}, \sigma - e, \text{count} + x) & \text{if count} + x < 2 \\ (\text{"active"}, 0, 0) & \text{otherwise} \end{cases}$$

$\delta_{int}(\text{phase}, \sigma, \text{count}) = (\text{"passive"}, \infty, \text{count})$

$\lambda(\text{"active"}, \sigma, \text{count}) = 1$

$ta(\text{phase}, \sigma) = \sigma$

DEVS examples (Cont'd)

- processor
  - simple workflow situation
    - connecting a generator to a processor
    - generator outputs jobs and processor takes some time to do them

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

where

Slide 13

$X = R$

$Y = R$

$S = \{\text{"passive"}, \text{"active"}\} \times R_0^+ \times R$

$\delta_{ext}(\text{phase}, \sigma, \text{job}, e, x) =$

$$\begin{cases} (\text{"active"}, \text{processing\_time}, \text{job}) & \text{if phase} = \text{"passive"} \\ (\text{phase}, \sigma - e, job) & \text{otherwise} \end{cases}$$

$\delta_{int}(\text{phase}, \sigma, \text{job}) = (\text{"passive"}, \infty, \text{job})$

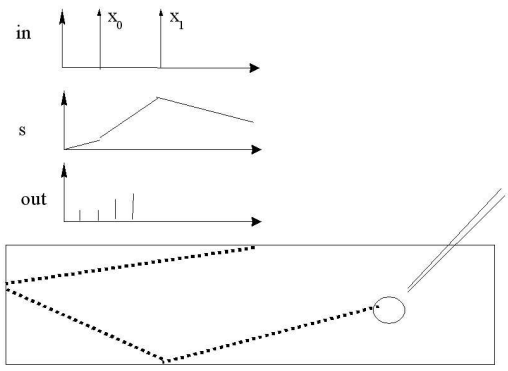
$\lambda(\text{"active"}, \sigma, \text{job}) = \text{job}$

$ta(\text{phase}, \sigma, \text{job}) = \sigma$

DEVS examples (Cont'd)

Slide 14

- ramp
  - DEVS can model systems whose discrete event nature is not immediately apparent
  - consider a billiard ball



DEVS examples (Cont'd)

Slide 15

- struck by a cue (external event)
- hitting the side of the table is considered as another input

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

where

$X = R$

$Y = R$

$S = \{\text{"passive","active"}\} \times R_0^+ \times R \times R$

δ_{ext}(phase, σ, input, position, e, x) = ("active", $\sigma - e$, x, position + e * input)

δ_{int}(phase, σ, input, position) =
("active", step_time, input, position + σ * input)

λ(phase, σ, input, position) = position + σ * input

ta(phase, σ, input, position) = σ

Classic DEVS with ports

Slide 16

- ports
  - modeling is made easier with the introduction of input and output ports
  - the more concrete DEVS formalism with port specifications

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

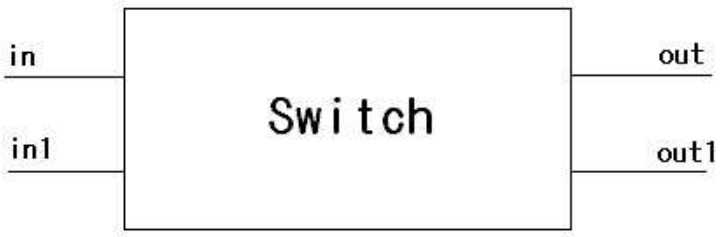
    where

    $X = \{(p, v) | p \in Inports, v \in X_p\}$ is the set of input ports and values
    $Y = \{(p, v) | p \in Outports, v \in Y_p\}$ is the set of output ports and values
    S is the set of *sequential* states
    $\delta_{ext} : Q \times X \rightarrow S$ is the *external state transition function*
    $\delta_{int} : S \rightarrow S$ is the *internal state transition function*
    $\lambda : S \rightarrow Y$ is the output function
    $ta : S \rightarrow R^{+}_{0,\infty}$ is the *time advance function*
    $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the set of *total states*

Classic DEVS with ports (Cont'd)

Slide 17

- switch



Classic DEVS with ports (Cont'd)

Slide 18

- operations
  - in standard position
    - jobs arriving on port "in" are sent out on port "out"
    - jobs arriving on port "in1" are sent out on port "out1"
  - in other position
    - jobs arriving on port "in" are sent out on port "out1"
    - jobs arriving on port "in1" are sent out on port "out"
  - switch controls the position and the polarity is toggled between true and false at each input

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

where

$Inports = \{"in", "in1"\}$, where $X_{in} = X_{in1} = V$ (an arbitrary set)
$X = \{(p, v) | p \in InPorts, v \in X_p\}$ is the set of input ports and values
$OutPorts = \{"out", "out1"\}$, where $Y_{out} = Y_{out1} = V$ (an arbitrary set)
$Y = \{(p, v) | p \in OutPorts, v \in Y_p\}$ is the set of output ports and values
$S = \{$"passive","busy"$\} \times R_0^+ \times \{in, in1\} \times V \times \{true, false\}$

Classic DEVS with ports (Cont'd)

Slide 19

δ_{ext}(phase, σ, inport, store, Sw, e, (p, v)) =

$$\begin{cases} (\text{"busy"}, \text{processing\_time}, p, v, \text{!Sw}) & \text{if phase} = \text{"passive" and } p \in \{in, in1\} \\ (\text{phase}, \sigma - e, \text{inport,store,Sw}) & \text{otherwise} \end{cases}$$

δ_{int}(phase, σ, inport, store, Sw, e, (p, v)) = ("passive", ∞, inport, store, Sw)

λ(phase, σ, inport, store, Sw) =

$$\begin{cases} (\text{out, store}) & \text{if phase} = \text{"busy" and Sw} = true \text{ and inport} = \text{in} \\ (\text{out1, store}) & \text{if phase} = \text{"busy" and Sw} = true \text{ and inport} = \text{in1} \\ (\text{out1, store}) & \text{if phase} = \text{"busy" and Sw} = false \text{ and inport} = \text{in} \\ (\text{out, store}) & \text{if phase} = \text{"busy" and Sw} = false \text{ and inport} = \text{in1} \end{cases}$$

ta(phase, σ, inport, store, Sw) = σ

Classic DEVS coupled models

Slide 20

- build models from components

  $N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select)$,

  where

  $X = \{(p, v) | p \in IPorts, v \in X_p\}$ is the set of input ports and values

  $Y = \{(p, v) | p \in OPorts, v \in Y_p\}$ is the set of output ports and values

  D is the set of the component names

  * components are DEVS models, for each $d \in D$

    $M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ is a DEVS with

    $X_d = \{(p, v) | p \in Iports, v \in X_p\}$ and $Y_d = \{(p, v) | p \in Oports, v \in Y_p\}$
  * external input coupling connects external inputs to component inputs

    $EIC \subseteq \{((N, ip_N), (d, ip_d)) | ip_N \in Iports, d \in D, ip_d \in Iports_d\}$
  * external output coupling connects component outputs to external outputs

    $EOC \subseteq \{((d, op_d), (N, op_N)) | op_N \in Oports, d \in D, op_d \in Oports_d\}$
  * internal coupling connects component outputs to component inputs

    $IC \subseteq \{((a, op_a), (b, ip_b)) | a, b \in D, op_a \in Oports_a, ip_b \in Iports_b\}$
    - no direct feedback loops are allowed for same component

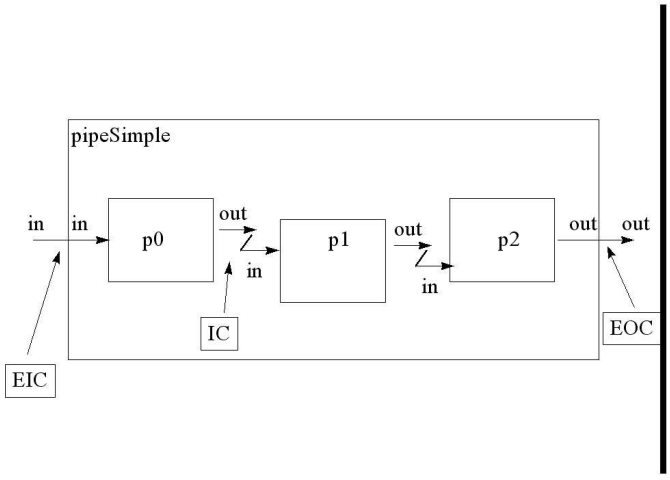
      $((d, op_d), (e, ip_d)) \in IC$ implies $d \neq e$

Classic DEVS coupled models (Cont'd)

$Select : 2^D - \{\} \rightarrow D$, the tie-breaking function

- simple pipeline

Slide 21



Classic DEVS coupled models (Cont'd)

Slide 22

$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select)$,

where

$Inports = \{"in"\}$

$X_{in} = V$ (an arbitrary)

$X = \{("in", v) | v \in V\}$

$Outports = \{"out"\}$

$Y_{out} = V$

$Y = \{("out", v) | v \in V\}$

$D = \{processor0, processor1, processor2\}$

$M_{processor2} = M_{processor1} = M_{processor0} = Processor$

$EIC = \{((N, "in"), (processor0, "in")\}$

$EOC = \{((processor2, "out"), (N, "out")\}$

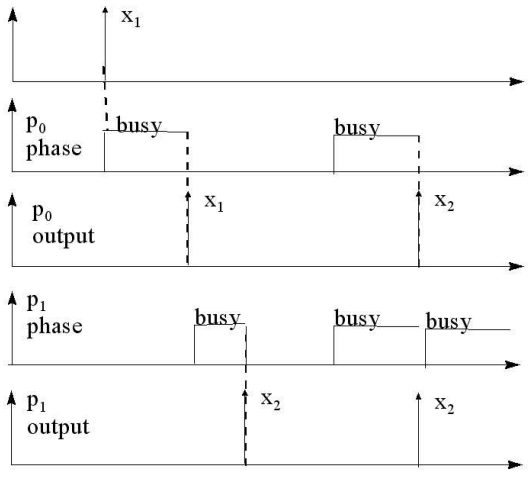
$IC = \{((processor0, "out"), (processor1, "in")),$
$((processor1, "out"), (processor2, "in"))\}$

$Select(D)$= the processor in D with the highest index

Classic DEVS coupled models (Cont'd)

Slide 23

– simultaneous events



Classic DEVS coupled models (Cont'd)

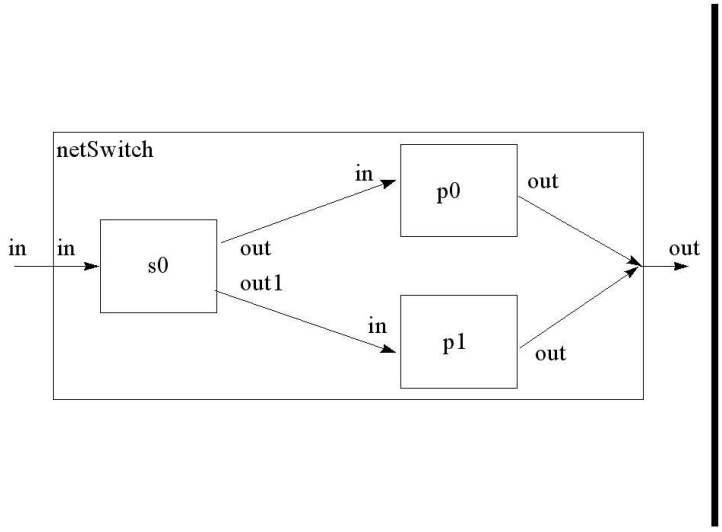
Slide 24

* processor0 and processor1 have same internal transition time
  a) processor1 applies its internal transition function first, transit to "passive" phase, and receives the input that processor0 produces
  b) processor0 applies its internal transition function first, outputs to processor 1, but processor 1 cannot receive the input because it is in "busy" phase
  - *Select* function specifies that if processor0 and processor1 are imminent, then processor1 is first

Classic DEVS coupled models (Cont'd)

Slide 25

- switch network



Classic DEVS coupled models (Cont'd)

Slide 26

$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select)$,

where

$Inports = \{"in"\}$
$X_{in} = V$ (an arbitrary)
$X_M = \{("in", v) | v \in V\}$
$Outports = \{"out"\}$
$Y_{out} = V$
$Y_M = \{("out", v) | v \in V\}$
$D = \{Switch0, processor0, processor1\}$
$M_{Switch0} = Switch; M_{processor1} = M_{processor0} = Processor$
$EIC = \{((N, "in"), (Switch0, "in")\}$
$EOC = \{((processor0, "out"), (N, "out")),$
$((processor1, "out"), (N, "out"))\}$
$IC = \{((Switch0, "out"), (processor0, "in")),$
$((Switch0, "out1"), (processor1, "in"))\}$

Parallel DEVS system specification

Slide 27

- parallel DEVS
  - differ from classic DEVS in allowing all imminent components

    $DEVS = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

    where

    $X_M = \{(p, v) | p \in Iports, v \in X_p\}$ is the set of input ports and values
    $Y_M = \{(p, v) | p \in Oports, v \in Y_p\}$ is the set of output ports and values
    S is the set of sequential states
    $\delta_{ext} : Q \times X_M^b \to S$ is the *external state transition* function
    $\delta_{int} : S \to S$ is the *internal state transition* function
    $\delta_{con} : Q \times X_M^b \to S$ is the *confluent transition* function
    $\lambda : S \to Y^b$ is the output function
    $ta : S \to R_{0,\infty}^+$ is the *time advance* function
    $Q = \{(s, e) | s \in S, 0 \le e \le ta(s)\}$ is the set of *total states*
  - differences
    * have a bag of inputs and additional transition function, called *confluent*
    * it is used when collision between external and internal events occurs

Parallel DEVS system specification (Cont'd)

Slide 28

- processor with buffer

$DEVS_{processing\_time} = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

where

$Iports = \{"in"\}$, where $X_{in} = V$ (an arbitrary set)

$X_M = \{(p, v) | p \in Iports, v \in X_p\}$ is the set of input ports and values

$Oports = \{"out"\}$, where $Y_{out} = V$ (an arbitrary set)

$Y_M = \{(p, v) | p \in Oports, v \in Y_p\}$ is the set of output ports and values

$S = \{"\text{passive}","\text{busy}"\} \times R_0^+ \times V^+$

$\delta_{ext}("\text{phase}", \sigma, \text{q}, e, (("in", x1), ("in", x2), \ldots, ("in", xn))) =$

$$\begin{cases} ("\text{busy}", \text{processing\_time}, q.\text{x1,x2,...,xn}) & \text{if phase} = "\text{passive}" \\ (\text{phase}, \sigma - e, q.\text{x1,x2,...,xn}) & \text{otherwise} \end{cases}$$

$\delta_{int}(\text{phase}, \sigma, \text{q.v}) = ("\text{passive}", \text{processing\_time}, \text{q})$

$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int}(s), 0, x)$

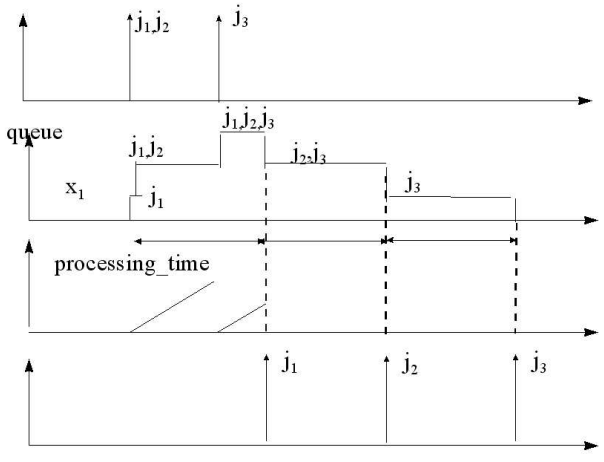
$\lambda("\text{busy}", \sigma, \text{q.v}) = v$

$ta(\text{phase}, \sigma, \text{q}) = \sigma$

Parallel DEVS system specification (Cont'd)

Slide 29

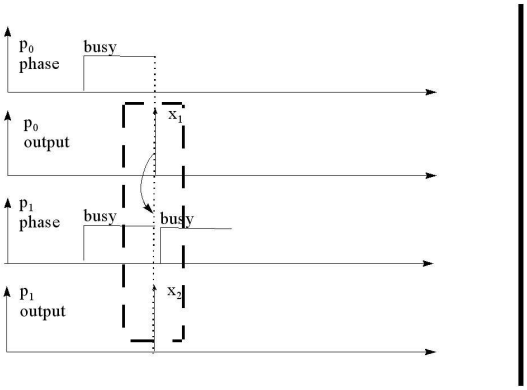
- a situation
  - * two jobs, j_1, j_2 arrive simultaneously



Parallel DEVS coupled models

- differences
  - specified in the same way as in classic DEVS except that the Select function is omitted
  - its semantics are much different
- simple pipeline (parallel DEVS)

Slide 30



Parallel DEVS coupled models (Cont'd)

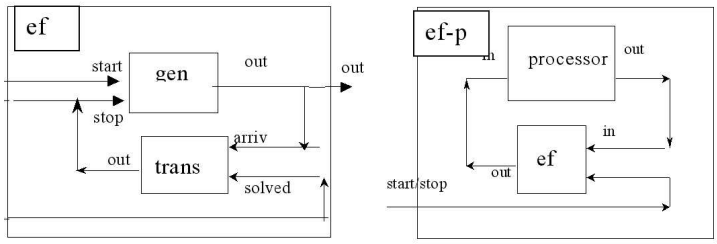
Slide 31

- processor0 and processor1 are imminent
- in classic DEVS only one would be chosen to execute
- in parallel DEVS
  - they both generate their outputs
  - processor0's output goes to processor1's input
  - since processor1 has both internal and external events, the confluent transition function is applied
  - by confluent function, internal transition function is applied first

Hierarchical models

Slide 32

- hierarchical models
  - coupled models with components that may be atomic or coupled models



  - encapsulate a generator and a transducer into a coupled model called an experimental frame, ef
  - the ef can then be coupled to a processor to provide it with a stream of inputs (from the generator) and observe the statistics of the processor's operation (the transducer)

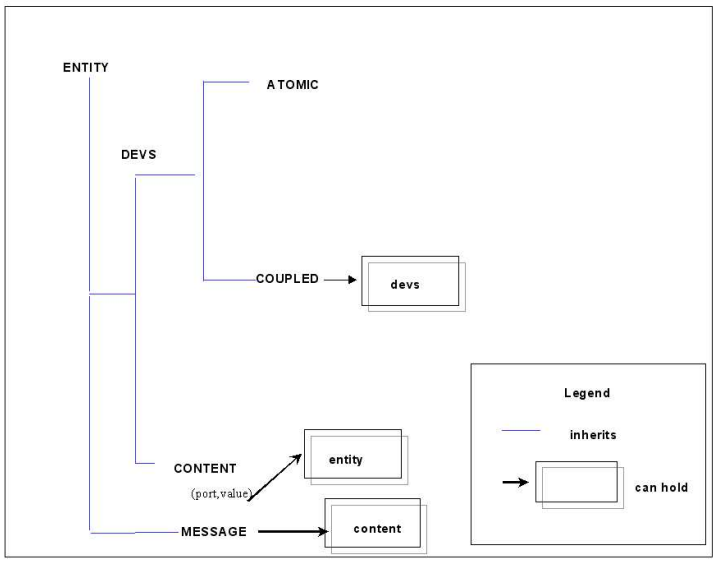
Object-oriented implementation of DEVS: an introduction

Slide 33

- introduction
  - DEVS is most naturally implemented in an object-oriented framework
  - the basic class is *entity*, from which class *devs* is derived
  - *devs* is specialized into classes *atomic* and *coupled*
  - class *content* derived from *entity* carry *ports* and their *values*
  - class *message* in the parallel DEVS are containers of *content* instances

Object-oriented implementation of DEVS: an introduction (Cont'd)

Slide 34



Object-oriented implementation of DEVS: an introduction (Cont'd)

– a simple processor as a derived class of atomic in DEVSJAVA

Slide 35

Class Processor *derived from* Atomic

external transition function

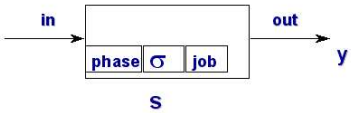
```
void  deltext(timetype e,message x)
{
  Continue();
  if (phase_is("passive"))
     for (int i=0; i< x.get_length();i++)
        if (message_on_port(x,"in",i)) {
          job = x.get_val_on_port("in",i);
          hold_in("busy",processing_time);
        }
}
```

output function

```
message out( )
{
  if (phase_is("busy"))
  message m = new message();
  entity val = job;
    m.add(make_content("out",val));
  return m;
}
```

internal transition function

```
void  deltint( )
{
   passivate();
}
```



Object-oriented implementation of DEVS: an introduction (Cont'd)

– an experimental frame class as a derived class of coupled

Slide 36