Devslang and DEVS operational semantics

Ernesto Posse

25th August 2004

McGill MSDL

Outline

- ✘ Introduction
- ✘ Devslang
- ✘ Formal operational semantics
- ✘ Future work

Introduction

- ✘ DEVS: “Discrete EVent System specification formalism”
- ✘ A formalism for modelling and simulating timed, discrete-event, composite, reactive/interactive systems.

Introduction

✘ Timed: A system “runs” over continuous time

Introduction

- ✘ Timed: A system “runs” over continuous time
- ✘ Discrete-event: In any given closed time-interval, only a finite number of events occur

Introduction

- ✘ Timed: A system “runs” over continuous time
- ✘ Discrete-event: In any given closed time-interval, only a finite number of events occur
- ✘ Composite: a system can be a collection of interconnected subsytems.

Introduction

- ✘ Timed: A system “runs” over continuous time
- ✘ Discrete-event: In any given closed time-interval, only a finite number of events occur
- ✘ Composite: a system can be a collection of interconnected subsytems.
- ✘ Reactive: a system can always react to external stimuli

Introduction

- ✘ Timed: A system “runs” over continuous time
- ✘ Discrete-event: In any given closed time-interval, only a finite number of events occur
- ✘ Composite: a system can be a collection of interconnected subsytems.
- ✘ Reactive: a system can always react to external stimuli
- ✘ Interactive: a system interacts with its environment (or components interact with each other)

DEVS

- Two types of DEVS components:
  - Atomic (or behavioural)
  - Coupled (or structural)

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*
  - $\delta^{int} : S \rightarrow S$ is an *internal transition function*

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*
  - $\delta^{int} : S \rightarrow S$ is an *internal transition function*
  - $\lambda : S \rightarrow Y \cup \{\perp\}$ is an *output function*

DEVS

- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*
  - $\delta^{int} : S \rightarrow S$ is an *internal transition function*
  - $\lambda : S \rightarrow Y \cup \{\perp\}$ is an *output function*
  - $\tau : S \rightarrow \mathbb{R}^+ \cup \{0, \infty\}$ is a *time-advance function*

DEVS

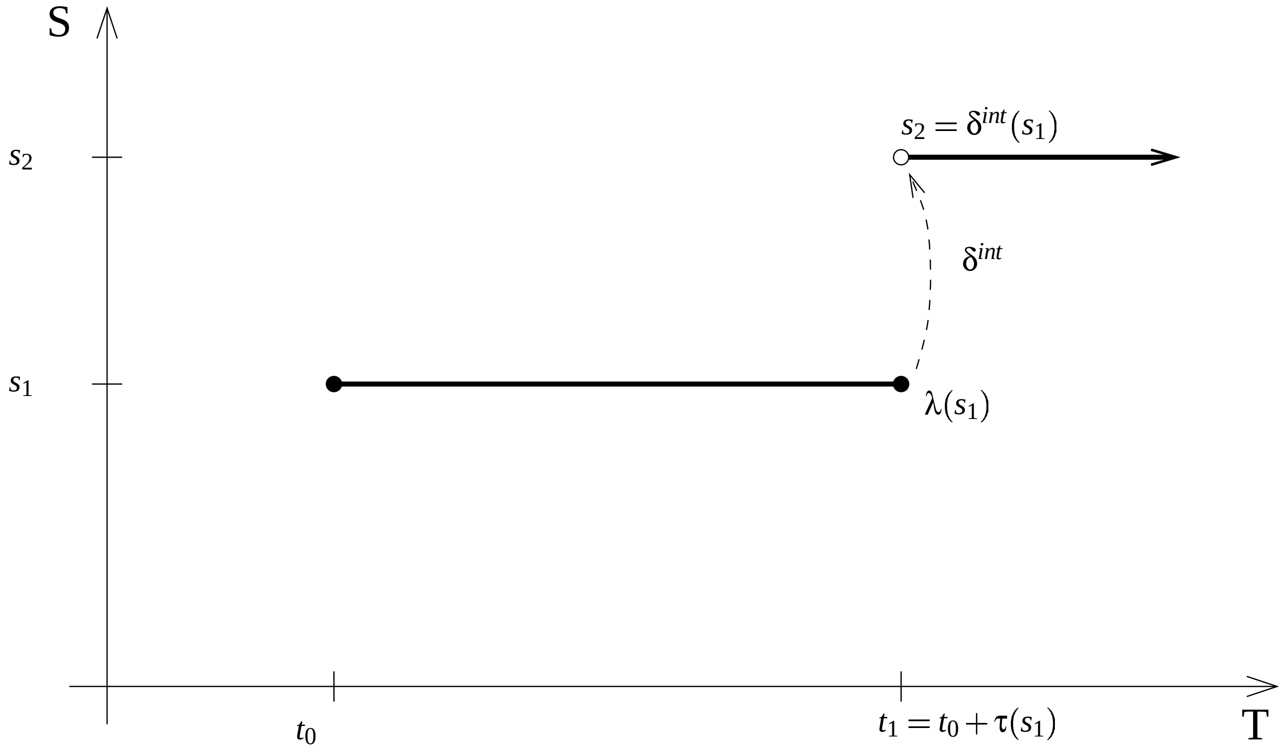
- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*
  - $\delta^{int} : S \rightarrow S$ is an *internal transition function*
  - $\lambda : S \rightarrow Y \cup \{\perp\}$ is an *output function*
  - $\tau : S \rightarrow \mathbb{R}^+ \cup \{0, \infty\}$ is a *time-advance function*
  - $\delta^{ext} : Q \times X \rightarrow S$ is an *external transition function,* where $Q \stackrel{def}{=} \{(s, e) \mid s \in S \,\text{and}\, 0 \leq e \leq \tau(s)\}$

DEVS

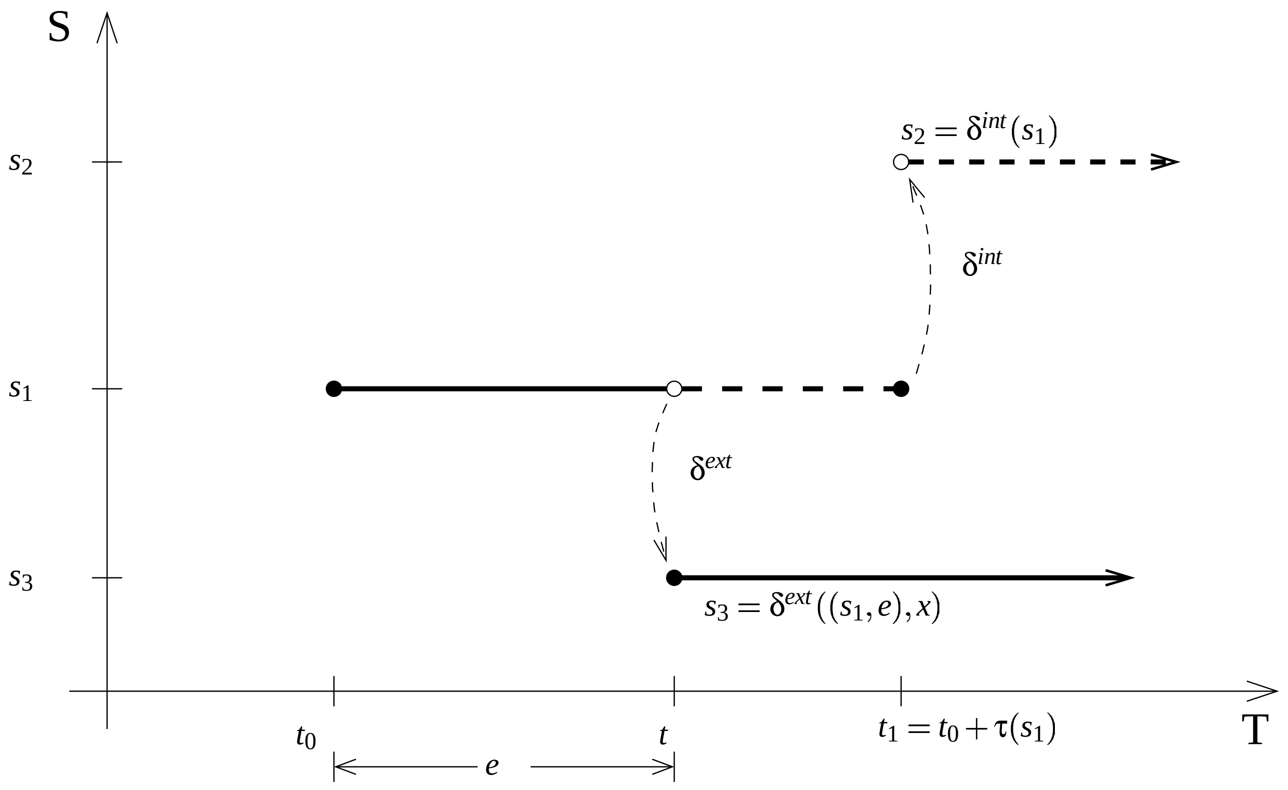
- An *atomic DEVS component* is a tuple $(X, Y, S, \delta^{int}, \delta^{ext}, \lambda, \tau, s_0)$ where:
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - S is a (possibly uncountable) set of *states*
  - $\delta^{int} : S \to S$ is an *internal transition function*
  - $\lambda : S \to Y \cup \{\perp\}$ is an *output function*
  - $\tau : S \to \mathbb{R}^+ \cup \{0, \infty\}$ is a *time-advance function*
  - $\delta^{ext} : Q \times X \to S$ is an *external transition function,* where $Q \stackrel{def}{=} \{(s, e) \mid s \in S \,\text{and}\, 0 \leq e \leq \tau(s)\}$
  - $s_0 \in S$ is an *initial state*

McGill MSDL

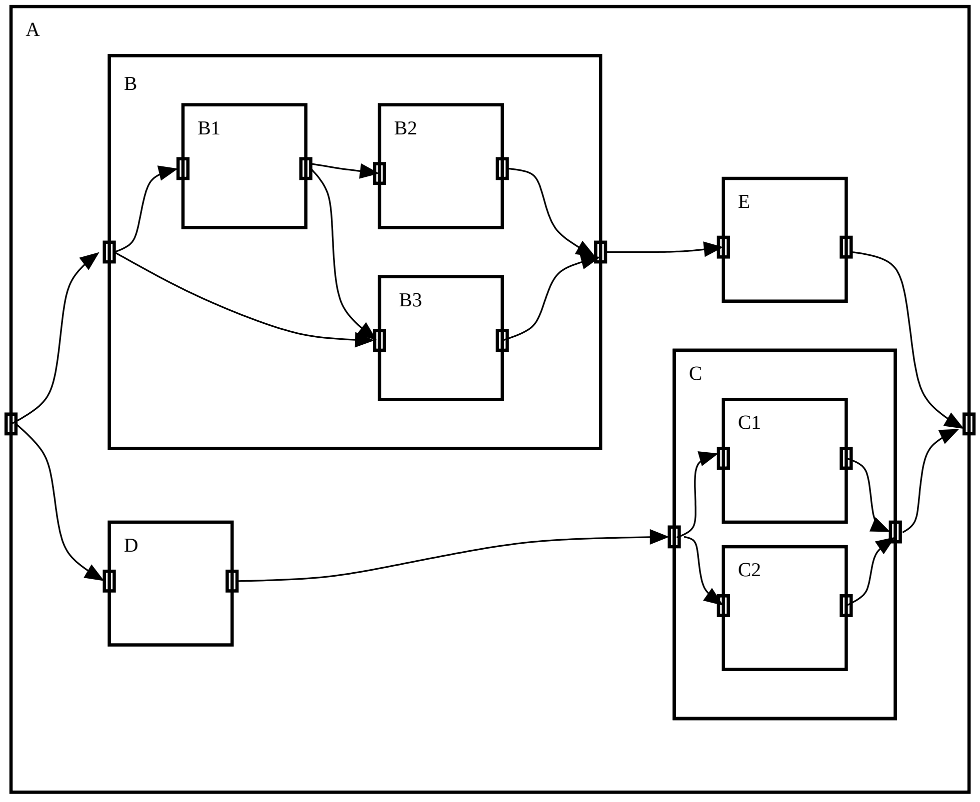
DEVS



DEVS


S
s_2
s_1
s_3
$s_2 = \delta^{int}(s_1)$
δ^{int}
δ^{ext}
$s_3 = \delta^{ext}((s_1, e), x)$
t_0
t
$t_1 = t_0 + \tau(s_1)$
T
e

DEVS



DEVS

- A *coupled DEVS component* is a tuple $(X, Y, N, C, infl, Z, sel)$ where
  - X is a set of possible *input values*
  - Y is a set of possible *output values*
  - N is a set of *component names*
  - C is a set of *components* (atomic or coupled) indexed by N
  - $infl : N \to 2^N$ is an *influencer function*
  - Z is a family of *transfer functions:*

$$
\begin{aligned}
Z &\subseteq \{Z_{i,j} : Y_i \to X_j | i, j \in N \text{ and } i \in infl(j)\} \\
&\cup \{Z_{\mathsf{self},k} : X \to X_k | \mathsf{self} \in infl(k)\} \\
&\cup \{Z_{k,\mathsf{self}} : Y_k \to Y | k \in infl(\mathsf{self})\}
\end{aligned}
$$

  - $sel : 2^N \to N$ is a *selection function*

McGill MSDL

Devslang

- Devslang is a language to represent DEVS models
- We need some representation for DEVS components:
  - ...to exchange models between different DEVS simulators
  - ...to be able to describe DEVS models in a more user-friendly fashion
  - ...to serve as the target representation for models in other formalisms
  - ...to take advantage of compiler technologies to generate efficient simulators

Devslang

- Components:

```
component Name(parameters) =
  inports a,b,c
  outports d,e
  ...
end
```

Devslang

- Atomic components:

```
component Name(parameters) =
  inports a,b,c
  outports d,e
  atomic
    ...
  end
end
```

Devslang

- Coupled components:

```
component Name(parameters) =
  inports a,b,c
  outports d,e
  coupled
    ...
  end
end
```

Devslang

- Atomic components:

```
atomic
  mode-definition-1
  ...
  mode-definition-n
  initial mode-invocation
end
```

Devslang

- Mode definitions:

```
mode name1(params1) =
  ...
end
```

- Mode invocation

```
name1(args)
```

Devslang

- Mode definitions:

```
mode name1(params1) =
  external-transitions
  after time-expr -> mode-invocation
  out output-record
end
```

Devslang

- Mode definitions:

```
mode name1(params1) =
  condition-1 -> mode-invocation-1,
  ...
  condition-n -> mode-invocation-n
  after time-expr -> mode-invocation
  out output-record
end
```

Devslang

- Variables that can be used in expressions:
  - input port names
  - parameters (mode and component)
  - `elapsed`
  - `infinity`

Devslang: Example 1

```
component Generator(period,value) =
  inports none
  outports y
  atomic
    mode active(next) =
      after next -> active(period)
      out {y: value}
    end
    initial active(period)
  end
end
```

Devslang: Example 1

```
component Generator(period,value) =
  inports x
  outports y
  atomic
    mode active(next) =
      any -> active(next - elapsed)
      after next -> active(period)
      out {y: value}
    end
    initial active(period)
  end
end
```

Devslang

- Configuration: $(state, time)$
- Event: $\mathsf{int}(t, v)$ or $\mathsf{ext}(t, v)$
- Trace of execution: Sequence of configurations

Devslang: Example 1

A = Generator(2,“a”)

State	Last trans	Event
active(2)	0	
		int(2,“a”)
active(2)	2	
		int(4,“a”)
active(2)	4	
		ext(4.5,x)
active(1.5)	4.5	
		int(6,“a”)
active(2)	6	
...		

Devslang: Example 2

```
component Store(response_time) =
  inports x
  outports y
  atomic
    mode receiving(next, data) =
      x = ("put", value) -> receiving(next-elapsed, value)
      x = "get"          -> responding(response_time, data)
      any                 -> receiving(next-elapsed, data)
      after infinity -> any
       out nothing
     end

    -- continues below
```

```
    mode responding(next, data) =
      any -> responding(next - elapsed, data)
      after next -> receiving(infinity, data)
      out {y: data}
   end
    initial receiving(infinity, nothing)
  end
end
```

Devslang: Example 3

```
component Processor(response_time, function) =
  inports x
  outports y
  atomic
    mode receiving(next) =
      any          -> busy(response_time, x)
      after next -> receiving(response_time)
      out nothing
     end

    -- continues below
```

```
     mode busy(next, job) =
       any -> busy(next - elapsed, job)
       after next -> receiving(response_time)
       out {y: function(job)}
    end
     initial receiving(response_time)
  end
end
```

Devslang

- Atomic components:

```
coupled
  component-instantiation-1
  ...
  component-instantiation-n
  connections
    connection-1
    ...
    connection-m
  select expr
end
```

Devslang

- Component instantiation:

  instance-name **=** *component-name***(***arguments***)**

  or

  instance-name **=** *component-definition*

- Connection

  from *outport* **to** *inport* **trans** *expr*

Devslang: Example 4

```
component SimpleCoupled(function) =
  inports none
  outports y
  coupled
    G = Generator(1.0,“a”)
    P = Processor(2.5,function)
  connections
    from G.y to P.x trans G.y + “b”
    from P.y to y trans P.y
  select P
  end
end
```

Formal operational semantics

- We want a semantics for Devslang and DEVS itself which is...
  - *abstract*: independent of specific simulation algorithms and engines, and for which we can apply *formal methods*
  - ...but not too abstract: close enough to the general idea of simulation/execution.

Formal operational semantics

- Labelled transition systems (LTS)!
- A *labelled transition system* is a tuple $(S, A, \rightarrow)$ where:
  - S is a set of states
  - A is a set of labels, representing actions, conditions or events
  - $\rightarrow \subseteq S \times A \times S$ is a *transition relation.* We write $s \xrightarrow{a} s'$ to mean $(s, a, s') \in \rightarrow$
- LTS are not FSA!

Formal operational semantics

- Each DEVS component A determines an LTS $\mathcal{M}(A) = (\mathbf{Configs}_A, \mathbf{Evts}_A, \rightarrow_A)$ where
  - $\mathbf{Configs}_A$ is the set of all A-configurations of the form (s,t)
  - $\mathbf{Evts}_A$ is the set of all A-events of the form $\mathtt{int}(t,v)$ or $\mathrm{ext}(t,v)$

Formal operational semantics

- ...and (for atomic components) $\rightarrow_A$ is the relation which satisfies:
  - Internal transitions (AIT): $(s, t_l) \xrightarrow{\mathsf{int}(t, \lambda(s))}_A (\delta^{int}(s), t)$ if $t = t_l + \tau(s)$
  - External transitions (AET): $(s, t_l) \xrightarrow{\mathsf{ext}(t, x)}_A (\delta^{ext}((s, t - t_l), x), t)$ if $t \leq t_l + \tau(s)$

McGill MSDL

Formal operational semantics

- ...and (for coupled components) $\rightarrow_A$ is the relation which satisfies:

- External transition (CET): $(\rho, t_l) \xrightarrow{\mathsf{ext}(t,x)}_B (\rho', t)$ if

  1. for each $n \in N$ such that self $\in infl(n)$ and $x_n \neq \perp$, $\rho(n) \xrightarrow{\mathsf{ext}(t,x_n)}_n \rho'(n)$, where $x_n \stackrel{def}{=} Z_{\mathsf{self},n}(x)$,
  2. and for all $n \in N$ such that self $\notin infl(n)$ or $x_n = \perp$, $\rho(n) = \rho'(n)$, where $x_n \stackrel{def}{=} Z_{\mathsf{self},n}(x)$

Formal operational semantics

...and

- Internal transition (CIT): $(\rho, t_l) \xrightarrow{\mathsf{int}(t,y)}_B (\rho', t)$ if

1. $\rho(i^*) \xrightarrow{\mathsf{int}(t,y^*)}_{i^*} \rho'(i^*)$,
2. for each $n \in N$ such that $i^* \in infl(n)$ and $n \neq \mathsf{self}$, $\rho(n) \xrightarrow{\mathsf{ext}(t,x_n)}_n \rho'(n)$ where $x_n = Z_{i^*,n}(y^*)$,
3. for all $n \in N$ such that $n \neq i^*$ and $i^* \notin infl(n)$, $\rho(n) = \rho'(n)$,
4. and $y = Z_{i^*,\mathsf{self}}(y^*)$ if $i^* \in infl(\mathsf{self})$ or $y = \perp$ if $i^* \notin infl(\mathsf{self})$

- where $i^* = sel(imm(\rho))$, and $imm(\rho)$ is the set of *imminent components* that is, of components which have a minimal time-to-next-transition.

Formal operational semantics

- *Behavioural equivalence*: having the “same” behaviour (bisimilarity)
- If A and B are behaviourally equivalent, then
  - an observer should not be able to distinguish between them...
  - ...therefore we should be able to replace one by the other in any context
- An equivalence relation $\sim$ is called a *congruence* if it is preserved by all contexts:
  - If $A \sim B$ then $C[A] \sim C[B]$ for all contexts $C[-]$

Formal operational semantics

- *Compositionality*: the meaning of a system is determined only by the meaning of its parts
- Why is compositionality important:
  - Simplicity of semantics
  - Efficiency of execution, simulation, analysis, optimization (example: separate compilation)

Formal operational semantics

- An operational semantics is compositional w.r.t. a behavioural equivalence, if the equivalence is a congruence
- If $A \sim B$ but $C[A] \not\sim C[B]$ then the meaning of $C[-]$ is not determined only by its parts

Formal operational semantics

Theorem. *Strong bisimilarity is a congruence for DEVS*

Future work

- ✘ Devslang interpreter/simulator
- ✘ Types
- ✘ Fully-abstract semantics
- ✘ Possible application of model-checking techniques
- ✘ Statecharts-to-DEVS transformation
- ✘ Variable-structure systems