

# Modelado y Simulación de Sistemas Dinámicos con el Formalismo DEVS

## Simulación de Modelos DEVS

Depto. Computación - Fac. Cs. Exactas Fco-Qcas y Naturales - UNRC

# Organización de la Presentación

- 1 Algoritmo de Simulación DEVS
- 2 Software de Modelado y Simulación con DEVS
- 3 PowerDEVS

# Organización de la Presentación

- 1 Algoritmo de Simulación DEVS
- 2 Software de Modelado y Simulación con DEVS
- 3 PowerDEVS

# Simulación de DEVS: Algunas Consideraciones

Para simular un modelo DEVS tendremos en cuenta lo siguiente:

- En las simulaciones, las acciones del **resto del universo** sobre el modelo en cuestión se representan mediante **fuentes**.
- En el caso de DEVS, dichas fuentes provocarán **secuencias de eventos** por lo que se podrán representar mediante modelos DEVS.
- Al conectarse los modelos DEVS de las fuentes y el sistema, tendremos un modelo **DEVS acoplado** que no tendrá entradas ni salidas.
- Por lo tanto, siempre simularemos un modelo DEVS acoplado que en el nivel jerárquico más alto no tendrá puertos de entrada ni de salida.

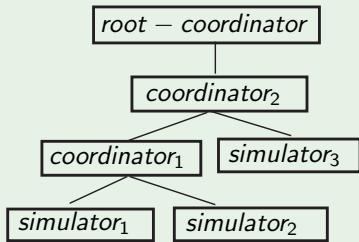
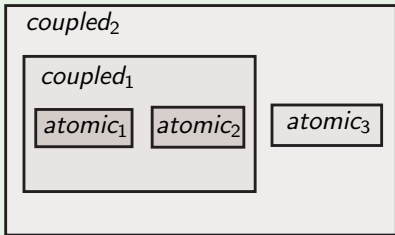
# Idea Básica del Algoritmo de Simulación DEVS

Dado un modelo acoplado  $N$  con submodelos  $d \in D$  (supondremos por ahora que son todos atómicos), podemos esbozar el siguiente algoritmo:

- 1 Comenzamos con  $t = t_0$  (tiempo inicial de simulación) y para cada  $d \in D$  evaluamos  $tn_d = t + ta_d(s_d) - e_d$  (tiempo del próximo evento del modelo  $d$ ).
- 2 Llamamos  $d^*$  al modelo  $d$  que tiene el mínimo  $tn_d$ .
- 3 Avanzamos el tiempo de simulación haciendo  $t = tn_{d^*}$ .
- 4 Calculamos el evento de salida  $y_{d^*} = \lambda_{d^*}(s_{d^*})$  y lo propagamos recalculando los estados  $s_d = \delta_{\text{ext}d}(s_d, e_d, y_{d^*})$  de los modelos  $d \in D$  que de acuerdo a  $IC$  deban recibir dicho evento y recalculamos los  $tn_d$ .
- 5 Calculamos el nuevo estado de  $d^*$  según  $s_{d^*} = \delta_{\text{int}}(s_{d^*})$  y recalculamos  $tn_{d^*}$ .
- 6 Volvemos al paso 2.

# Estructura del Algoritmo de Simulación DEVS

Una manera de implementar el algoritmo es mediante una estructura de objetos que replique la estructura del modelo, comunicada mediante **mensajes**:



## Estructura del Algoritmo de Simulación DEVS

- A cada modelo **atómico** le asociamos un objeto de clase *simulator*. Cada *simulator* tendrá el estado del modelo atómico correspondiente  $s$ , sus funciones de transición, salida, etc. y una variable  $tn$  que calcula el tiempo de la siguiente transición interna de dicho modelo atómico.
- A cada modelo **acoplado** le asociamos un objeto de clase *coordinator*. Cada *coordinator* manejará la propagación de eventos de sus hijos (de clase *simulator* o *coordinator*) y calculará una variable  $tn$  que será igual al **mínimo** de los  $tn$  de sus hijos (y por lo tanto igual al tiempo de la próxima **transición interna** de dicho modelo acoplado).
- En el tope de la jerarquía habrá un objeto de la clase *root* – *coordinator*, que tendrá como único hijo un *coordinator* asociado al modelo acoplado completo. El *root* – *coordinator* será el encargado de avanzar el **tiempo de simulación**.

## Algoritmo de Simulación DEVS: Mensajes

La comunicación entre los padres (de clase *coordinator* o *root – coordinator*) y sus hijos (de clase *simulator* ó *coordinator*) se basa en los siguiente mensajes:

- Mensaje de **inicialización** (i-message) de padre a hijo (al **comienzo** de la simulación).
- Mensaje de **transición interna** (\*-message) de padre a hijo (cuando corresponda una transición interna al hijo).
- Mensaje de **transición externa** (x-message) de padre a hijo (cuando corresponda una transición externa al hijo)
- Mensaje de **salida** (y-message) de hijo a padre (cuando corresponda propagar un evento)



# Algoritmo de Simulación DEVS: *simulator*

DEVS-simulator

variables:

$tl$  // time of last event

$tn$  // time of next event

$s$  // state of the DEVS atomic model

$e$  // elapsed time in the actual state

$y = (y.value, y.port)$  // current output of the DEVS atomic model

when receive i-message ( $i, t$ ) at time  $t$

$tl = t - e$

$tn = tl + ta(s)$

when receive \*-message ( $*$ ,  $t$ ) at time  $t$

$y = \lambda(s)$

send y-message ( $y, t$ ) to parent coordinator

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

$tl = t$

$tn = t + ta(s)$

when receive x-message ( $x, t$ ) at time  $t$

$e = t - tl$

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

$tl = t$

$tn = t + ta(s)$

end DEVS-simulator

# Algoritmo de Simulación DEVS: *coordinator*

DEVS-coordinator

variables:

$tl$  // time of last event

$tn$  // time of next event

$y = (y.value, y.port)$  // current output of the DEVS coordinator

$D$  // list of children

$IC$  // list of connections of the form  $[(d_i, port_1), (d_j, port_2)]$

$EIC$  // list of connections of the form  $[(N, port_1), (d_j, port_2)]$

$EOC$  // list of connections of the form  $[(d_i, port_1), (N, port_2)]$

when receive i-message  $(i, t)$  at time  $t$

send i-message  $(i, t)$  to all the children

when receive \*-message  $(*, t)$  at time  $t$

send \*-message  $(*, t)$  to  $d^*$

$d^* = \arg[\min_{d \in D}(d.tn)]$

$tl = t$

$tn = t + d^*.tn$

(Continúa)

## Algoritmo de Simulación DEVS: *coordinator*

```
when receive x-message  $((x.value, x.port), t)$  at time  $t$ 
   $(v, p) = (x.value, x.port)$ 
  for each connection  $[(N, p), (d, q)]$ 
    send x-message  $((v, q), t)$  to child  $d$ 
   $d^* = \arg[\min_{d \in D}(d.tn)]$ 
   $tl = t$ 
   $tn = t + d^*.tn$ 
when receive y-message  $((y.value, y.port), t)$  from  $d^*$ 
  if a connection  $[(d^*, y.port), (N, q)]$  exists
    send y-message  $((y.value, q), t)$  to parent coordinator
  for each connection  $[(d^*, p), (d, q)]$ 
    send x-message  $((y.value, q), t)$  to child  $d$ 
end DEVS-coordinator
```

# Algoritmo de Simulación DEVS: *root* – *coordinator*

DEVS-root-coordinator

variables:

$t$  // global simulation time

$d$  // child (coordinator or simulator)

$t = t_0$

send i-message ( $i, t$ ) to  $d$

$t = d.tn$

loop

send \*-message ( $*, t$ ) to  $d$

$t = d.tn$

until end of simulation

end DEVS-root-coordinator

# Organización de la Presentación

- 1 Algoritmo de Simulación DEVS
- 2 Software de Modelado y Simulación con DEVS
- 3 PowerDEVS

# Software de Simulación de Modelos DEVS

Actualmente, existen varias herramientas de software basadas en DEVS, desarrollada por los distintos grupos de investigación que trabajan en el tema:

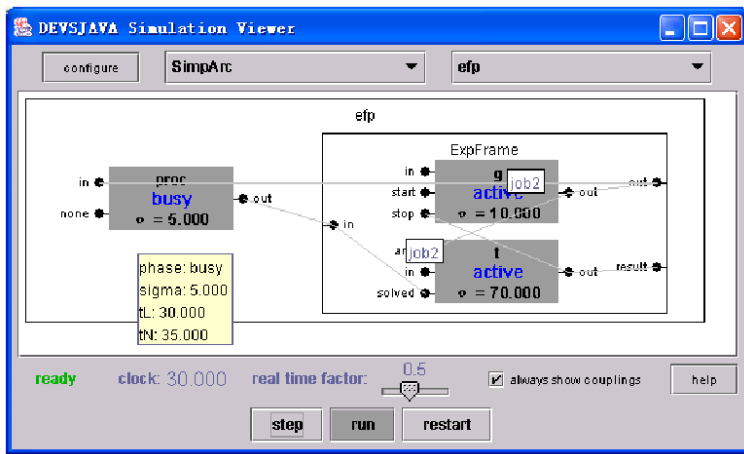
- ADEVS, DEVS/C++ y DEVSJAVA (University of Arizona).
- DEVSSim++ (KAIST, Corea).
- CD++ (Carleton University y UBA).
- LSIS-DME (LSIS, Marsella, Francia)
- VLE (Université du Littoral Côte d'Opale, Francia).
- SmallIDEVS (Brno University of Technology, República Checa).
- JDEVS (Université de Corse).
- **PowerDEVS** (Universidad Nacional de Rosario).

# DEVSJAVA

DEVSJAVA es una herramienta desarrollada por el grupo de Bernard Zeigler (University of Arizona):

- Es un simulador de propósito general, basado en el algoritmo descripto, completamente programado en **Java**.
- Cuenta con una **interfaz gráfica** para crear modelos y visualizar resultados.

# DEVSJAVA



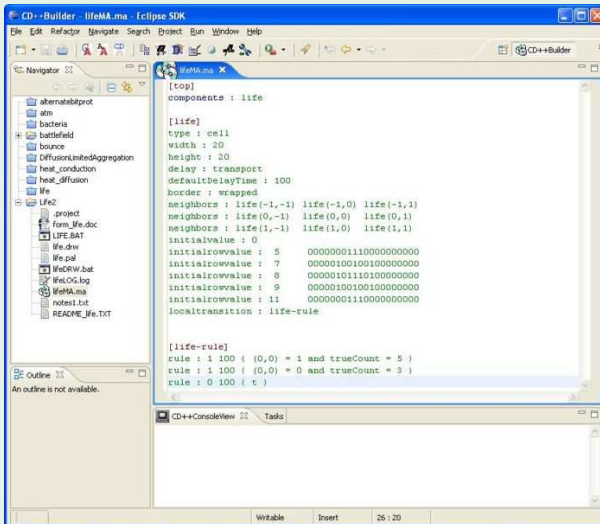


# CD++

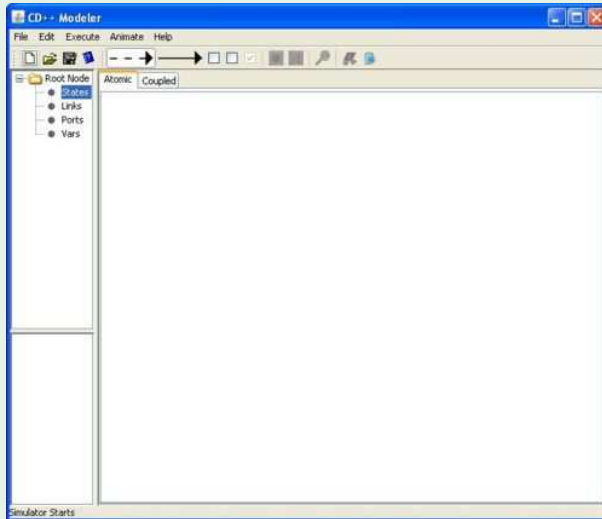
CD++ es una herramienta desarrollada por el grupo de Gabriel Wainer (Carleton University y Universidad de Buenos Aires), orientada a la implementación de **Cell-DEVS**.

- El motor de simulación está programado en **C++**.
- La interfaz de usuario para modelado está implementada en **Eclipse**.
- Cuenta con numerosas herramientas de visualización y animación de resultados de simulación para modelos celulares.

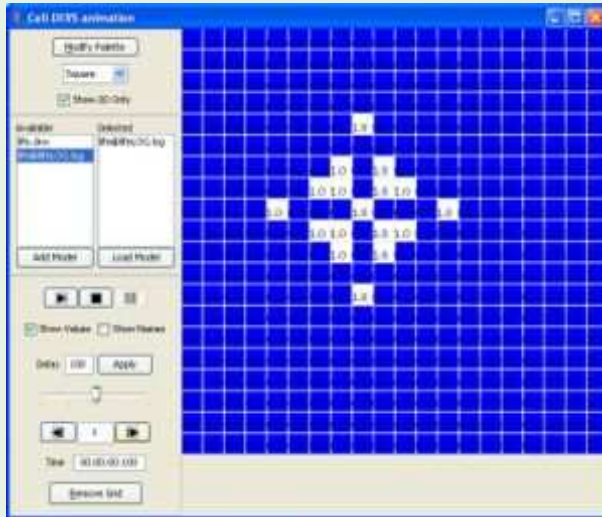
# CD++



# CD++



# CD++

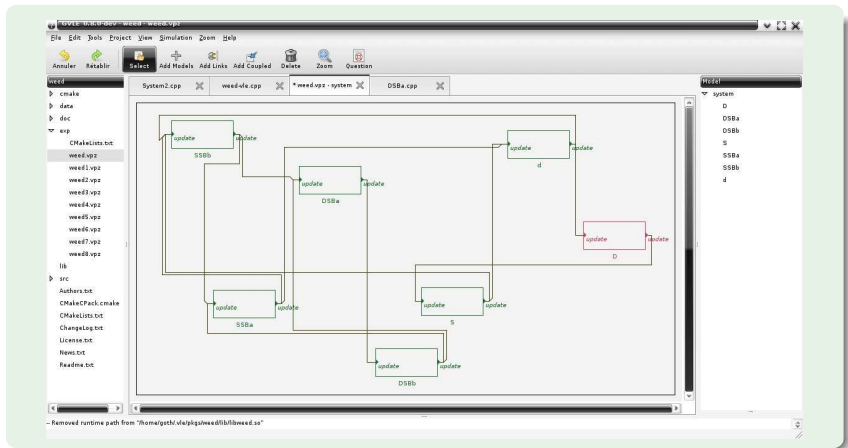


# VLE

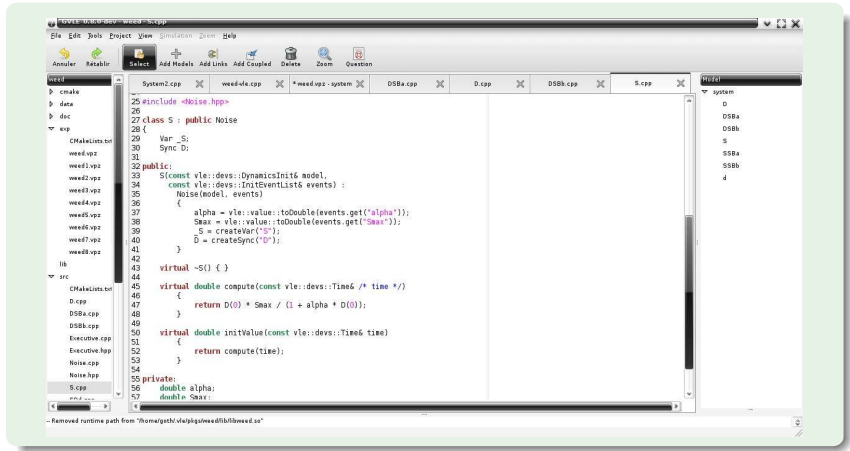
VLE (Virtual Laboratory Environment) es una herramienta nueva desarrollada por el grupo de Eric Ramat (Université du Littoral Côte d'Opale).

- El motor de simulación está programado en C++.
- Cuenta con una interfaz de usuario para el acoplamiento gráfico de modelos.
- Combina herramientas amigables para la edición, simulación y visualización de resultados.

# VLE



# VLE



# Organización de la Presentación

- 1 Algoritmo de Simulación DEVS
- 2 Software de Modelado y Simulación con DEVS
- 3 PowerDEVS

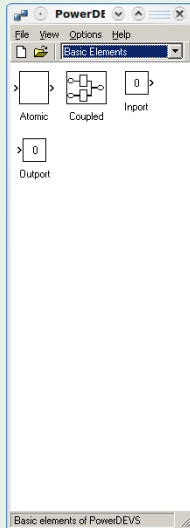


# PowerDEVS

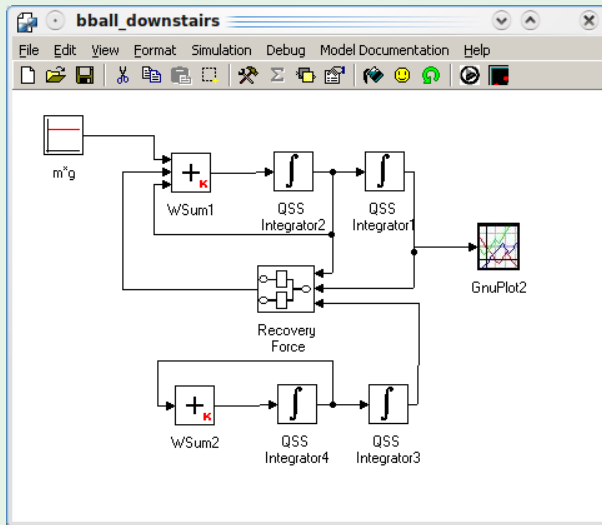
PowerDEVS es una herramienta desarrollada en la Universidad Nacional de Rosario, con las siguientes características:

- El motor está programado completamente en **C++**.
- Cuenta con herramientas **gráficas** de edición, simulación y visualización.
- Hay versiones para Windows, Linux y RTAI (tiempo real).
- Tiene librerías completas para simulación de **sistemas continuos** e híbridos.
- Tiene librerías para simulación y control en **tiempo real**.
- Está integrado con **Scilab**, lo que le provee herramientas de cálculo y procesamiento de datos avanzados.

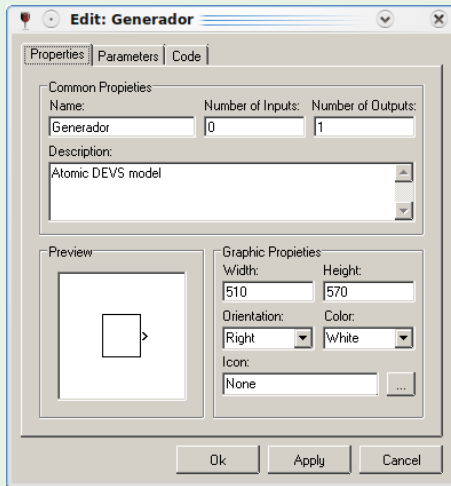
## PowerDEVS. Ventana de librerías



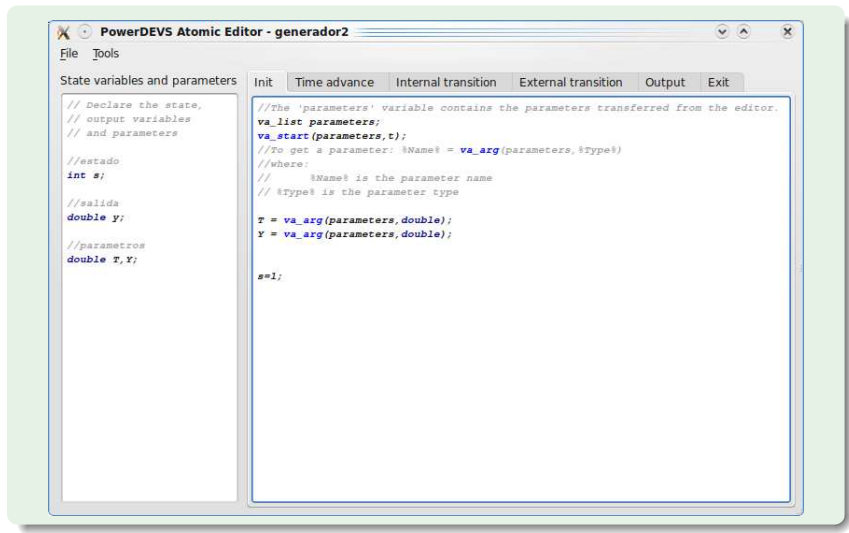
# PowerDEVS. Ventana de modelo



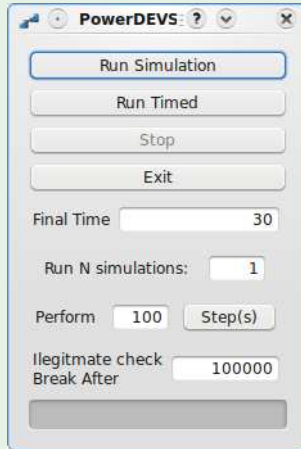
## PowerDEVS. Ventana de edición de bloques



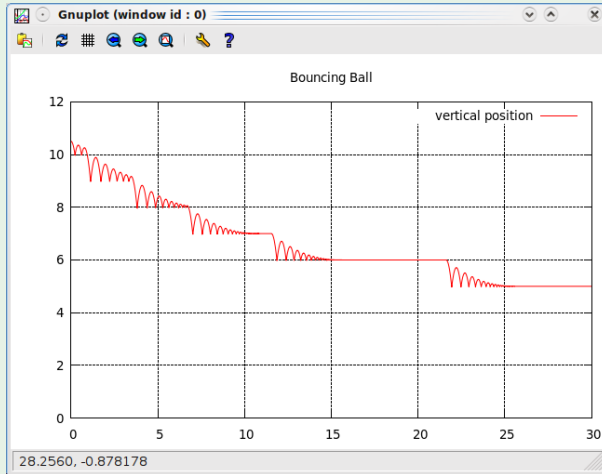
# PowerDEVS. Editor de modelos atómicos



# PowerDEVS. Ventana de Simulación



# PowerDEVS. Resultados de Simulación



## PowerDEVS. Ejemplo Modelo Atómico

El siguiente ejemplo muestra como se puede especificar un modelo atómico usando el **editor de modelos atómicos** de PowerDEVS:

$$P_2 = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

$$X = Y = \mathbb{R}^+$$

$$S = \mathbb{R}^+ \times \{true, false\}$$

$$\delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}((\sigma, busy), e, x) = \begin{cases} (\sigma - e, true) & \text{si } busy = true \\ (x, true) & \text{en otro caso} \end{cases}$$

$$\delta_{\text{int}}(s) = (\infty, false)$$

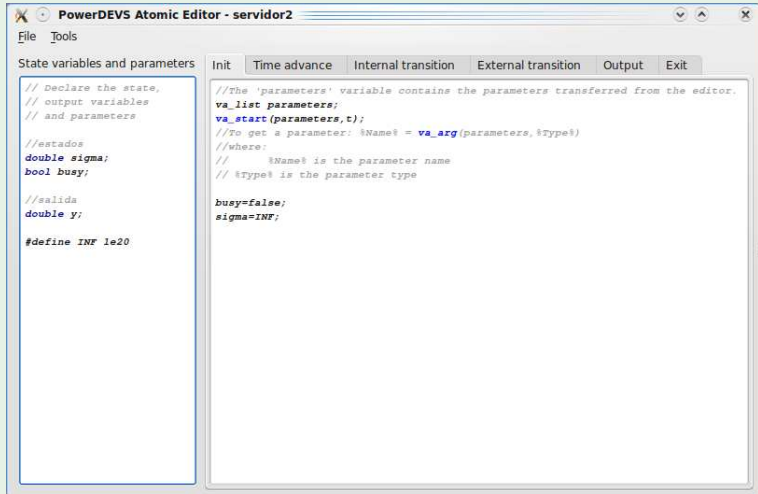
$$\lambda(s) = 1$$

$$ta((\sigma, busy)) = \sigma$$



# PowerDEVS. Ejemplo Modelo Atómico

## Declaraciones y Estado inicial



The screenshot shows the 'PowerDEVS Atomic Editor - servidor2' window. The 'Init' tab is selected, displaying the following code:

```
// Declare the state,
// output variables
// and parameters

//estados
double sigma;
bool busy;

//salida
double y;

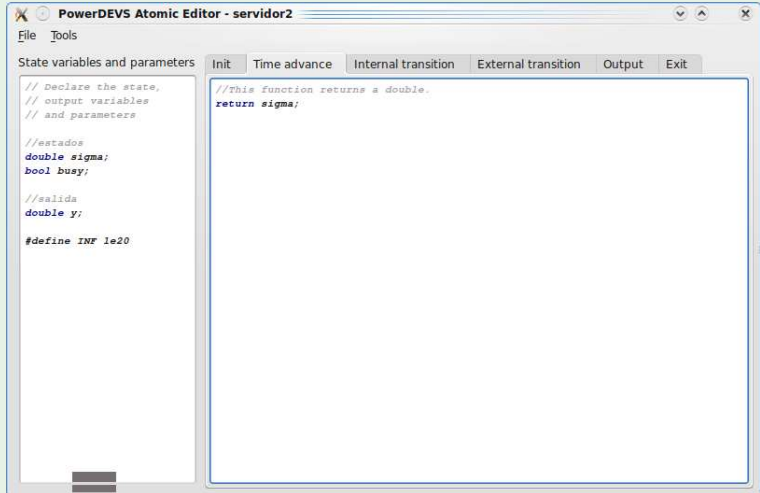
#define INF 1e20

//The 'parameters' variable contains the parameters transferred from the editor.
va_list parameters;
va_start(parameters,t);
//To get a parameter: %Name% = va_arg(parameters,%Type%)
//where:
//      %Name% is the parameter name
//      %Type% is the parameter type

busy=false;
sigma=INF;
```

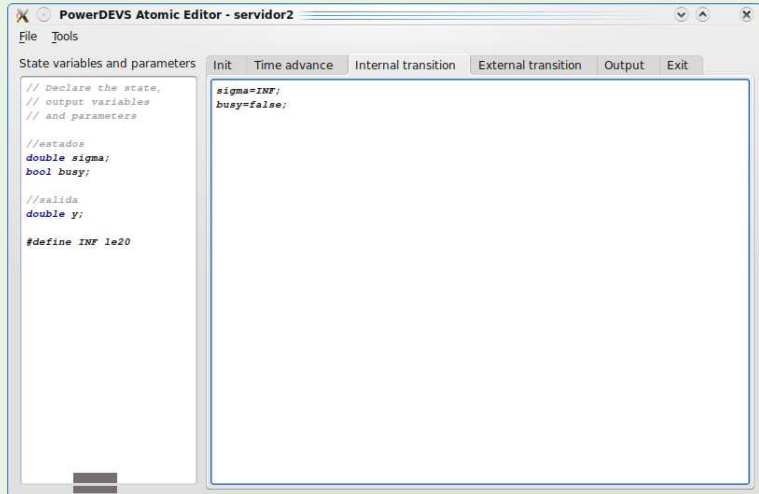
# PowerDEVS. Ejemplo Modelo Atómico

## Función de avance de tiempo



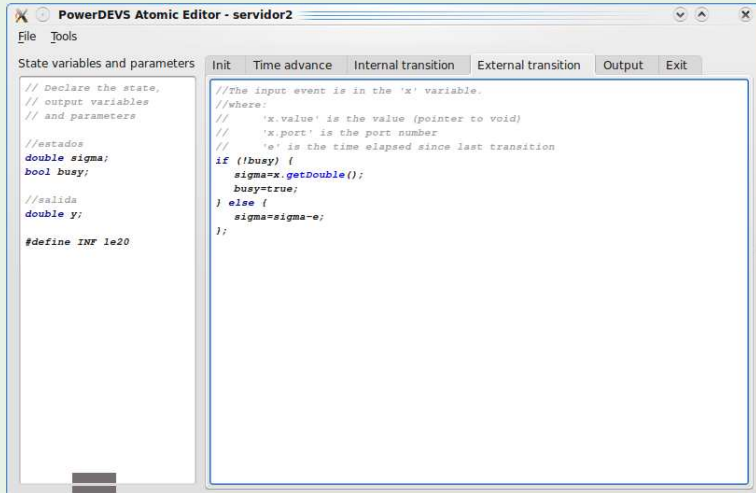
# PowerDEVS. Ejemplo Modelo Atómico

## Función de transición interna



# PowerDEVS. Ejemplo Modelo Atómico

## Función de transición externa



```
// Declare the state,
// output variables
// and parameters

//estados
double sigma;
bool busy;

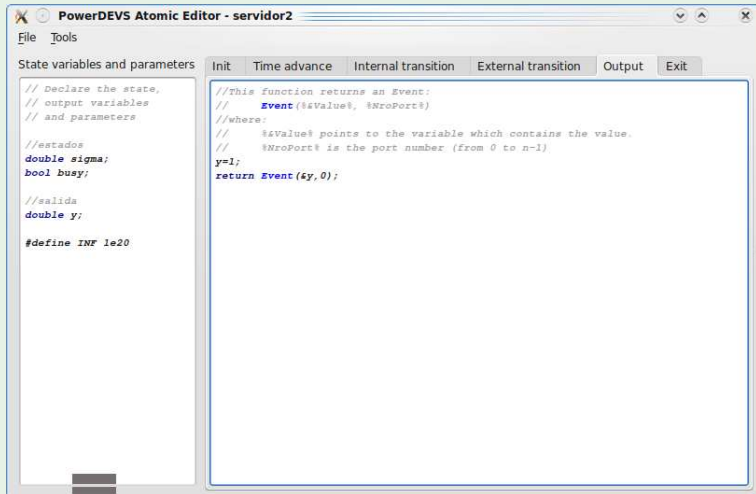
//salida
double y;

#define INF 1e20

//The input event is in the 'x' variable.
//where:
//  'x.value' is the value (pointer to void)
//  'x.port' is the port number
//  'e' is the time elapsed since last transition
if (!busy) {
    sigma=x.getDouble();
    busy=true;
} else {
    sigma=sigma-e;
};
```

# PowerDEVS. Ejemplo Modelo Atómico

## Función de salida



The screenshot shows the 'PowerDEVS Atomic Editor - servidor2' window. The 'Output' tab is selected, displaying the output function code. The left pane shows the state variables and parameters, and the right pane shows the output function code.

```
// Declare the state,  
// output variables  
// and parameters  
  
//estados  
double sigma;  
bool busy;  
  
//salida  
double y;  
  
#define INF 1e20
```

```
//This function returns an Event:  
//  
//      Event(%Value%, %NroPort%)  
//where:  
//      %Value% points to the variable which contains the value.  
//      %NroPort% is the port number (from 0 to n-1)  
y=1;  
return Event(&y,0);
```