Modelado y Simulación de Sistemas Dinámicos

Sistemas de Eventos Discretos

Ernesto Kofman

FCEIA - Universidad Nacional de Rosario.
CIFASIS – CONICET. Argentina

Organización de la Presentación

1. Representaciones Gráficas Tradicionales
   - Grafos de Transición de Estado
   - Redes de Petri
   - Limitaciones de los formalismos gráficos
2. Formalismo DEVS
   - Definición del Formalismo
   - Modelos DEVS Acoplados
   - Extensiones del Formalismo
3. Simulación de Modelos DEVS
   - Algoritmo de Simulación DEVS
   - Software de Modelado y Simulación con DEVS
   - PowerDEVS

Organización de la Presentación

Representaciones Gráficas de Eventos Discretos

Existen varios formalismos de representación gráfica de sistemas por eventos discretos:

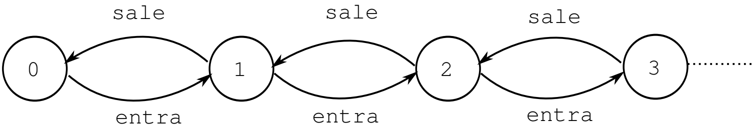
- Grafos de Transición de Estados
- Redes de Petri
- Grafcet

entre los más populares

Grafos de Transición de Estado

Ejemplo Introductorio

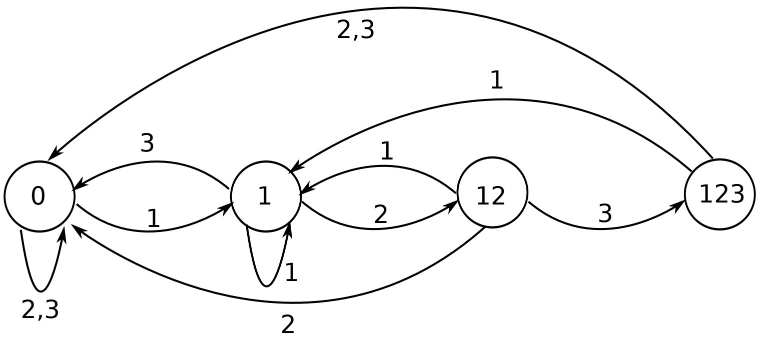
Consideremos un sistema que cuenta el número de personas que hay en una habitación sensando las personas que entran y salen.



- El estado del sistema es el número de personas en la habitación.
- Cada vez que entra o sale una persona, ocurre un evento de entrada y se produce una transición de estado.
- No hay una representación explícita del tiempo en este modelo.

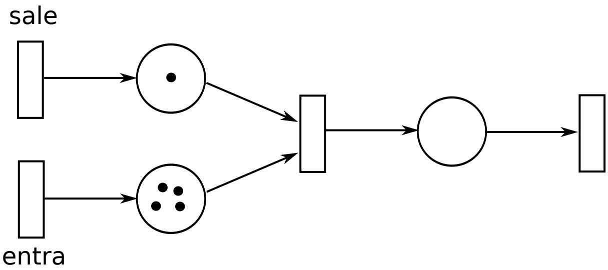
Grafos de Transición de Estados

Sistema de Reconocimiento de Secuencias



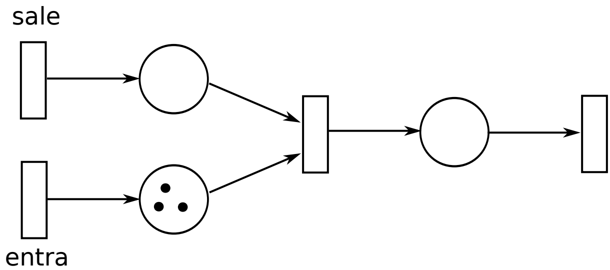
El sistema reconoce la secuencia $1, 2, 3$. El estado representa la secuencia reconocida hasta el momento actual.

Redes de Petri



- Las redes de Petri se componen de arcos, lugares (places), transiciones (transitions) y marcas (tokens).
- El estado queda determinado por el número de marcas en cada lugar.

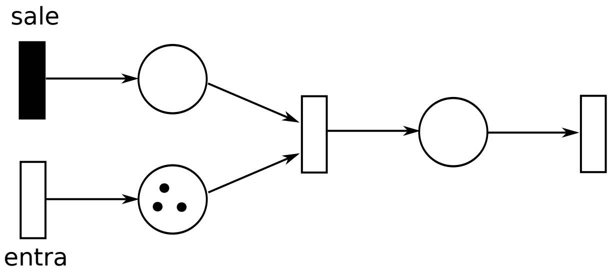
Redes de Petri
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

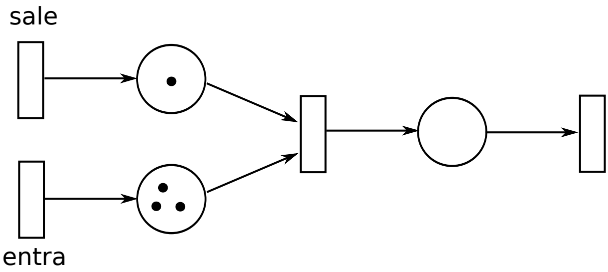
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

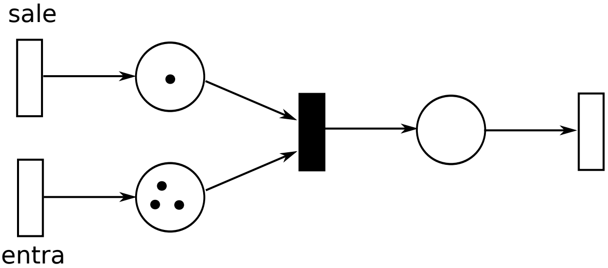
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

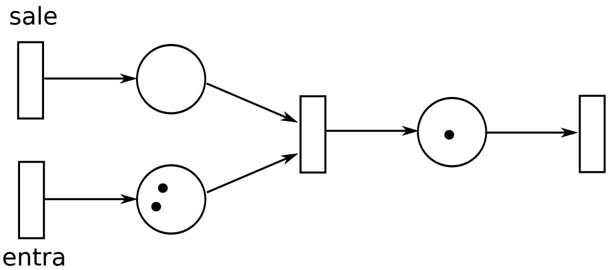
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

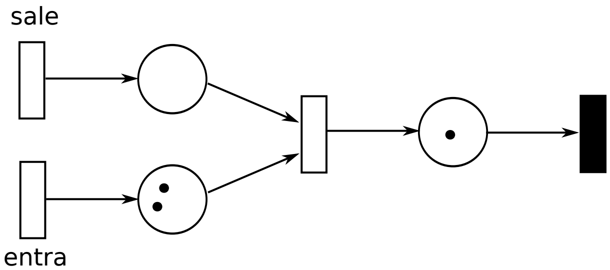
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

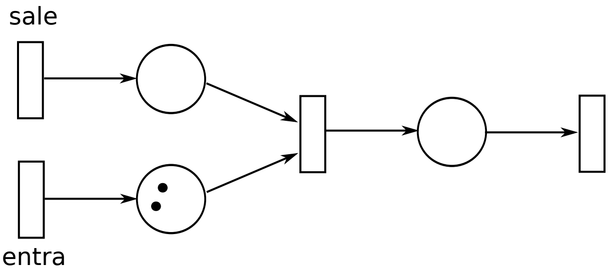
Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

Redes de Petri

Ejemplo Introductorio



- Las transiciones se activan cuando en todos los lugares precedentes hay al menos una marca. Tras esto, todos los lugares precedentes pierden una marca y los lugares posteriores ganan una.
- Las transiciones de entrada son aquellas que no tienen lugares precedentes y se activan por acciones externas.

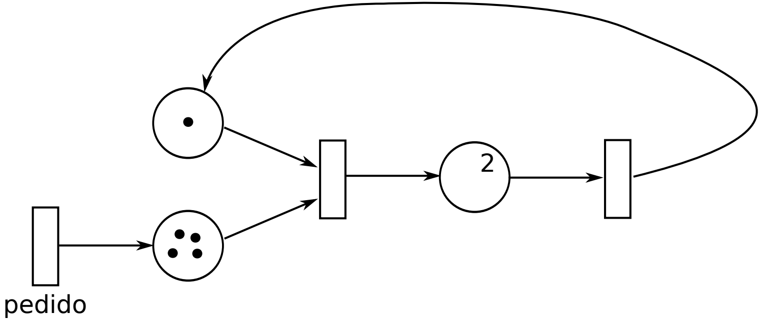
Redes de Petri Temporizadas

- Una de las maneras de incluir la acción del tiempo es considerar que las marcas deben permanecer un tiempo mínimo en un lugar para activar las transiciones correspondientes.
- De esta forma, la dinámica del sistema no depende sólo del ordenamiento de la secuencia de eventos de entrada, sino también de los tiempos de los eventos.
- Otra alternativa de temporización es considerar que los disparos de cada transición consumen cierto tiempo.

Redes de Petri Temporizadas

Ejemplo: Sistema cola-servidor

Un sistema cola-servidor recibe pedidos, los acumula y los procesa. El tiempo de procesamiento de cada pedido es de 2 segundos.



Redes de Petri

Software de Simulación

Hay varias alternativas para modelar y simular Redes de Petri:

- Herramientas gratuitas multiplataforma en Java: Pipe2, Tapaal, HiSim, etc.
- Diversas herramientas comerciales específicas.
- Librerías y Toolboxes de herramientas de software más generales (Modelica y PowerDEVS, entre ellas).

Limitaciones de los formalismos gráficos: Ejemplo

Un usuario presiona un pulsador repetidas veces y el sistema mide el tiempo entre los sucesivos intervalos. Si el último intervalo es más largo que el anterior, el sistema emite un sonido largo. En otro caso, emite un sonido corto.

- Los eventos de entrada son los pulsos que produce el usuario.
- Los eventos de salida toman dos valores (sonido corto o largo).
- El estado es el tiempo transcurrido entre los dos últimos pulsos de entrada. El conjunto de posibles estados es infinito (los números reales positivos).
- El número de cambios de estado en un intervalo de tiempo es finito, por lo que se trata de un sistema de eventos discretos.

Ninguno de los formalismos gráficos puede representar el comportamiento de este sistema.

Limitaciones de los formalismos gráficos

- El número de estados posibles es siempre finito.
- De manera similar, los posibles eventos de entrada pertenecen a un conjunto finito.
- Es muy difícil (o imposible a veces) reutilizar partes de modelos para construir modelos más complejos.
- En general no se pueden encapsular sub–modelos, lo que hace imposible armar modelos muy grandes.

Estas limitaciones motivaron en parte el desarrollo del formalismo DEVS para Modelado y Simulación de Sistemas de Eventos Discretos generales.

Organización de la Presentación

El Formalismo DEVS

El formalismo DEVS, formulado por Bernard Zeigler a mediados de los 70, permite representar cualquier sistema que tenga un número finito de cambios en un intervalo finito de tiempo.

Un modelo DEVS procesa una secuencia de eventos de entrada y de acuerdo a su condición inicial produce una secuencia de eventos de salida.

Trayectorias de Eventos

Evento

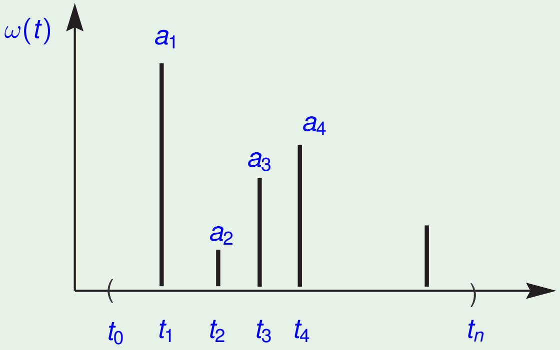
Un evento es la representación de un cambio instantáneo y puede caracterizarse por el valor que adopta y por el tiempo en el que ocurre.

Trayectoria de Eventos

Sea $\omega : (t_0, t_n) \to A \cup \{\phi\}$ una función donde (t_0, t_n) es un intervalo de los reales, A es un conjunto arbitrario y ϕ es un elemento que no pertenece a A y representa la ausencia de evento.
Luego, ω define una trayectoria de eventos si y sólo si existe un conjunto de puntos $t_1, t_2, \cdots, t_{n-1}$ con $t_i \in (t_0, t_n)$ tales que $\omega(t_i) \in A$ para $i = 1, \cdots, n-1$ y $\omega(t) = \phi$ para todo otro punto en (t_0, t_n).

Trayectorias de Eventos

Trayectoria de Eventos



DEVS – Modelo Atómico

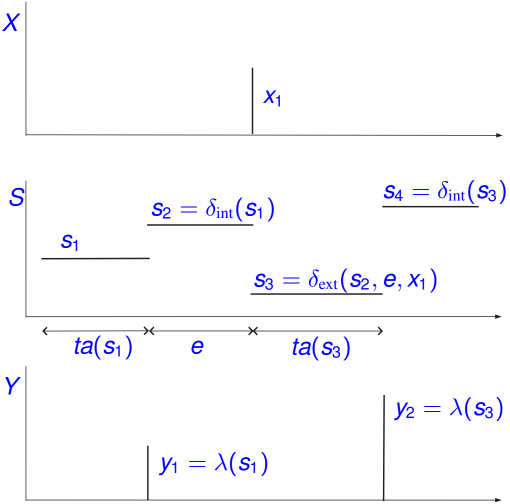
Un modelo atómico DEVS se define por la estructura:

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

donde:

- X es el conjunto de valores de entrada.
- Y es el conjunto de valores de salida.
- S es el conjunto de valores de estado.
- δ_{int}, δ_{ext}, λ y ta son las funciones que definen la dinámica del sistema.

DEVS – Trayectorias



DEVS – Ejemplo 1

Un sistema (que llamaremos generador) produce eventos que representan trabajos a realizar. El sistema produce eventos según la siguiente secuencia: $t = 0, y = 1$; $t = 1, y = 2$; $t = 3, y = 1$; $t = 4, y = 2$; etc.

$$G_1 = < X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta >$$

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

$$\delta_{\text{int}}(s) = 3 - s$$

$$\lambda(s) = 3 - s$$

$$ta(s) = s$$

DEVS – Ejemplo 2

Un procesador recibe trabajos identificados por un número real positivo que indica cuanto tiempo demora en procesarse dicho trabajo. Transcurrido el tiempo de procesamiento, el procesador emite un evento con valor 1.

$$P_1 = < X, Y, S, \delta_{\rm int}, \delta_{\rm ext}, \lambda, ta >$$
$$X = Y = S = \mathbb{R}^+$$
$$\delta_{\rm ext}(s, e, x) = x$$
$$\delta_{\rm int}(s) = \infty$$
$$\lambda(s) = 1$$
$$ta(s) = s$$

Este modelo se olvida del trabajo que estaba procesando al recibir uno nuevo

DEVS – Ejemplo 2

Para ignorar los eventos de entrada mientras se está procesando un trabajo, podemos modificar el modelo anterior como sigue:

$$P_2 = < X, Y, S, \delta_{\mathrm{int}}, \delta_{\mathrm{ext}}, \lambda, ta >$$

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

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

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

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

$$\lambda(s) = 1$$

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

DEVS – Ejemplo 3

Un usuario presiona un pulsador repetidas veces y el sistema mide el tiempo entre los sucesivos intervalos. Si el último intervalo es más largo que el anterior, se emite un evento con el valor *true*. En otro caso, emite un evento con valor *false*.

$$M_1 = < X, Y, S, \delta_{\rm int}, \delta_{\rm ext}, \lambda, ta >$$
$$X = \{\text{“pulse”}\}$$
$$Y = \{true, false\}$$
$$S = \mathbb{R}^+ \times \{true, false\} \times \mathbb{R}^+$$
$$\delta_{\rm ext}(s, e, x) = \delta_{\rm ext}((\tau, isLonger, \sigma), e, x) = (e, ?e > \tau, 0)$$
$$\delta_{\rm int}(s) = \delta_{\rm int}(\tau, isLonger, \sigma) = (\tau, isLonger, \infty)$$
$$\lambda(s) = \lambda(\tau, isLonger, \sigma) = isLonger$$
$$ta(s) = ta(\tau, isLonger, \sigma) = \sigma$$

Algunas consideraciones sobre los modelos DEVS

- Para facilitar la construcción de los modelos, se suele incluir en el estado una variable σ que sea igual al tiempo de avance. Es decir, $ta(s) = ta(\ldots, \sigma) = \sigma$.
- Los estados para los cuales $ta(s) = 0$ se denominan estados transitorios e indican que se producirá un evento de salida de manera inmediata.
- Los estados para los cuales $ta(s) = \infty$ se denominan estados pasivos e indican que no se producirá ningún cambio en el sistema mientras no llegue un evento de entrada.

DEVS – Legitimidad

El siguiente modelo representa un generador que produce eventos con valor 1.

$$G_2 =< X, Y, S, \delta_{\rm int}, \delta_{\rm ext}, \lambda, ta >$$
$$X = Y = S = \mathbb{R}^+$$
$$\delta_{\rm int}(s) = s/2$$
$$\lambda(s) = 1$$
$$ta(s) = s$$

A partir de un estado inicial $s = 1$ por ejemplo, al cabo de 2 segundos se habrán producido infinitos eventos de salida.

Se dice entonces que el modelo G_2 es ilegítimo.

DEVS – Legitimidad

Legitimidad

Un modelo DEVS es legítimo si sólo puede producir una cantidad acotada de cambios en cada intervalo finito de tiempo.

Para establecer formalmente la condición de legitimidad se definen las funciones δ^+ y Σ según:

$$\delta^+_{\text{int}}(s,0) \triangleq s;\;\; \delta^+_{\text{int}}(s,n+1) \triangleq \delta_{\text{int}}(\delta^+_{\text{int}}(s,n))$$

$$\Sigma(s,0) \triangleq ta(s);\;\; \Sigma(s,n) \triangleq \Sigma(s,n-1) + ta(\delta^+_{\text{int}}(s,n-1))$$

y un modelo DEVS resultará legítimo si y sólo si

$$\lim_{n\to\infty} \Sigma(s,n) \to \infty \qquad \forall s \in S$$

Acoplamiento de Modelos

Si bien cualquier sistema de eventos discretos puede teóricamente representarse mediante un modelo DEVS atómico, en la práctica obtener dicho modelo puede ser extremadamente difícil.

La tarea de modelización puede simplificarse enormemente con la idea del acoplamiento, es decir, pensando los sistemas como una composición de subsistemas más simples.

Acoplamiento mediante puertos de entrada y salida

Si bien hay varias alternativas para definir formalmente el acoplamiento de modelos DEVS, en este curso trabajaremos con el acoplamiento mediante puertos.

Puertos de Entrada y Salida

Diremos que un modelo atómico DEVS cuenta con m puertos de entrada y p puertos de salida cuando sus conjuntos de valores de entrada (X) y salida (Y) tengan la siguiente forma:

$$X = X_0 \times \{inp_0\} \cup \cdots \cup X_{m-1} \times inp_{m-1}$$
$$Y = Y_0 \times \{outp_0\} \cup \cdots \cup Y_{p-1} \times outp_{p-1}$$

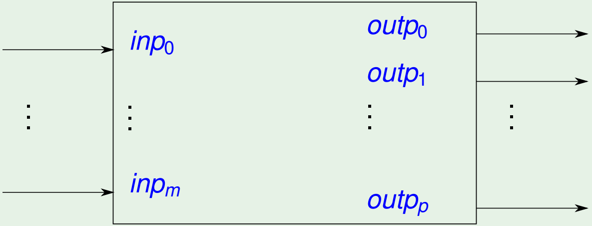
donde inp_j denota el j–ésimo puerto de entrada y $outp_j$ el j–ésimo puerto de salida.

Cada evento de entrada tendrá la forma $(x, port)$, donde $x \in X_{port}$ representa el valor del evento y $port \in \{inp_0, \ldots, inp_{m-1}\}$ indicará el puerto por el que ingresa dicho evento.

Cada evento de salida tendrá la forma $(y, port)$, donde $y \in Y_{port}$ representa el valor del evento y $port \in \{outp_0, \ldots, outp_{m-1}\}$ indicará el puerto por el que sale dicho evento.

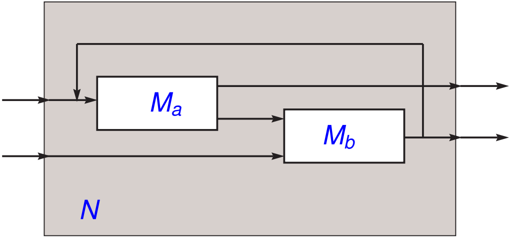
Puertos de Entrada y Salida – Representación gráfica

Representaremos los modelos DEVS atómicos con puertos mediante bloques como muestra la figura:



Acoplamiento de Modelos DEVS

En el acoplamiento modular los eventos de salida de un modelo DEVS se convierten en eventos de entrada de otro.



En el modelo DEVS acoplado N encontramos dos modelos atómicos (M_a y M_b) y distintas conexiones (entre M_a y M_b, entre N y M_a, entre N y M_b).

Modelos DEVS Acoplados – Definición

Un modelo DEVS acoplado se define mediante la estructura

$$N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select), \text{ donde:}$$

- El conjunto de valores de entrada del modelo acoplado N es $X_N = X_0 \times \{inp_0\} \cup \cdots \cup X_{m-1} \times inp_{m-1}$.
- El conjunto de valores de salida del modelo acoplado N es $Y_N = Y_0 \times \{outp_0\} \cup \cdots \cup Y_{p-1} \times outp_{p-1}$.
- D es el conjunto de referencias a componentes.
- Para cada $d \in D$, $M_d = (X_d, Y_d, S_d, \delta_{\mathrm{int}\,d}, \delta_{\mathrm{ext}\,d}, \lambda_d, ta_d)$ es un modelo DEVS con puertos.

Modelos DEVS Acoplados – Definición

Un modelo DEVS acoplado se define mediante la estructura

$$N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select), \text{ donde:}$$

- EIC (external input coupling) es el conjunto de conexiones desde las entradas de N hacia las entradas de los M_d. Cada elemento de EIC tiene la forma $[(N, inp_i), (d_k, inp_j)]$ (lo que indica una conexión desde la i–ésima entrada de N hacia la j–ésima entrada de d_k).
- EOC (external output coupling) es el conjunto de conexiones desde las salidas de los M_d hacia las salidas de N. Cada elemento de EOC tiene la forma $[(d_k, outp_j), (N, outp_i)]$ (lo que indica una conexión desde la j–ésima salida de d_k hacia la i–ésima salida de N).

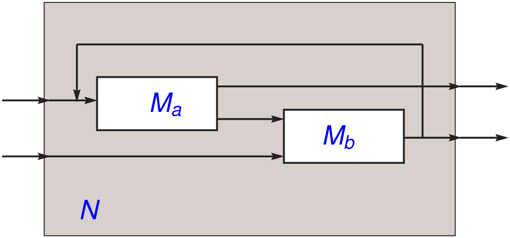
Modelos DEVS Acoplados – Definición

Un modelo DEVS acoplado se define mediante la estructura

$$N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select), \text{ donde:}$$

- IC (internal coupling) es el conjunto de conexiones desde las salidas de los M_d hacia las entradas de los M_d. Cada elemento de IC tiene la forma $[(d_l, outp_i), (d_k, inp_j)]$ (lo que indica una conexión desde la i–ésima salida de d_l hacia la j–ésima entrada de d_k). Debe cumplirse que $l \neq k$.
- Finalmente $Select : 2^D \to D$ es una función de desempate, que decide prioridades en caso de eventos simultáneos.

Modelos DEVS Acoplados. Ejemplo



- $EIC = \{[(N,0),(a,0)];[(N,1),(b,1)]\}$
- $EOC = \{[(a,0),(N,0)];[(b,0),(N,1)]\}$
- $IC = \{[(a,1),(b,0)];[(b,0),(a,0)]\}$

(De acá en más para denotar inp_j o $outp_j$ utilizaremos j).

Clausura bajo acoplamiento

Teorema: *Clausura bajo acoplamiento*

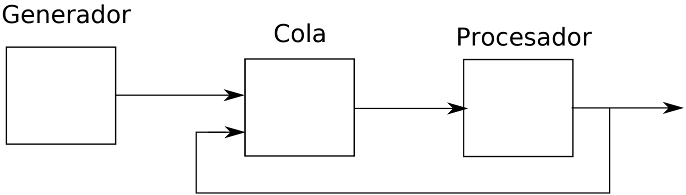
Dado un modelo DEVS acoplado $N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select)$ cualquiera, siempre existe un modelo DEVS atómico $M = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta)$ equivalente.

Esto es, M es tal que dada una condición incial arbitraria para los estados de los M_d, existe una condición inicial para el estado de M tal que para cualquier trayectoria de entrada las trayectorias de salida de N y M son idénticas.

La clausura bajo acoplamiento permite utilizar los modelos acoplados tratándolos como si fueran modelos atómicos, acoplándolos con otros modelos atómicos o acoplados.

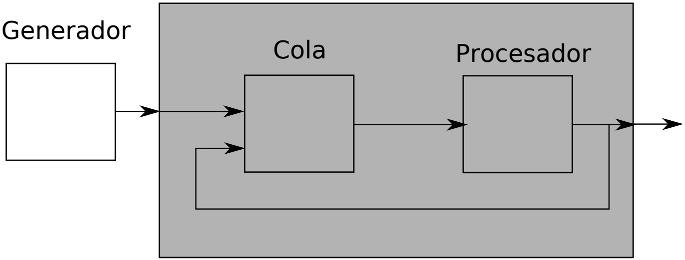
DEVS – Acoplamiento Jerárquico

La clausura bajo acoplamiento garantiza la posibilidad de realizar acoplamiento jerárquico, para construir fácilmente modelos complejos:



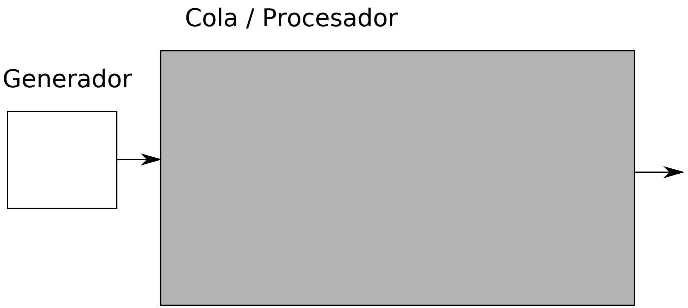
DEVS – Acoplamiento Jerárquico

La clausura bajo acoplamiento garantiza la posibilidad de realizar acoplamiento jerárquico, para construir fácilmente modelos complejos:



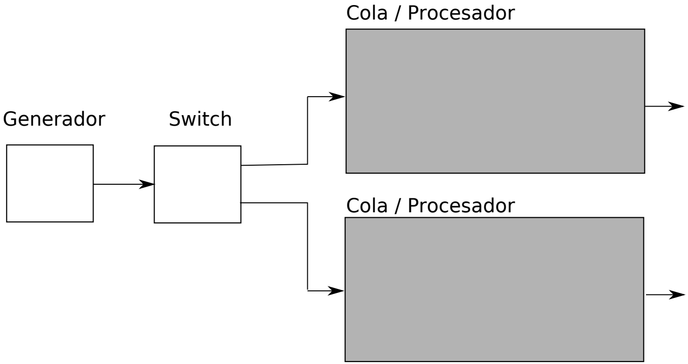
DEVS – Acoplamiento Jerárquico

La clausura bajo acoplamiento garantiza la posibilidad de realizar acoplamiento jerárquico, para construir fácilmente modelos complejos:



DEVS – Acoplamiento Jerárquico

La clausura bajo acoplamiento garantiza la posibilidad de realizar acoplamiento jerárquico, para construir fácilmente modelos complejos:



Eventos Simultáneos

Consideremos el siguiente generador (emite alternadamente 1 y 2 por el puerto 0 cada 1 segundo):

$G_3 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta)$ con:
$X = \{1, 2\}$; $Y = \{(1, 0), (2, 0)\}$; $\delta_{\text{int}}(s) = 3 - s$; $ta(s) = 1$; $\lambda(s) = (s, 0)$;

y el siguiente modelo de sumador (emite la suma de los últimos valores recibidos en cada puerto de entrada):

$$
\begin{aligned}
&S_1 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta) \text{ con:}\\
&X = \mathbb{R} \times \{0, 1\};\ Y = \mathbb{R} \times \{0\};\ S = \mathbb{R} \times \mathbb{R} \times \mathbb{R}^+\\
&\delta_{\text{int}}(s) = \delta_{\text{int}}(u_0, u_1, \sigma) = (u_0, u_1, \infty);\\
&\delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}((u_0, u_1, \sigma), e, (x_v, port)) = (\tilde{u}_0, \tilde{u}_1, 0);\\
&\lambda(s) = \lambda(u_0, u_1, \sigma) = (u_0 + u_1, 0);\quad ta(s) = \sigma;
\end{aligned}
$$

donde $\tilde{u}_j = u_{port}$ si $j = port$ y $\tilde{u}_j = u_j$ si $j \neq port$.

Eventos Simultáneos

Si acoplamos dos generadores G_{3_a} y G_{3_b} con el sumador S_1 y suponemos que en $t = 0$ ambos generadores tienen $s_a = s_b = 1$ y el sumador tiene $u_0 = u_1 = 0$, $\sigma = \infty$, al simular lo que ocurre, podemos obtener lo siguiente:

- $t = 1$: Transición interna de G_{3_a} con salida $y_a = 1$. Estados: $s_a = 2$, $s_b = 1$, $u_0 = 1$, $u_1 = 0$, $\sigma = 0$.
- $t = 1$: Transición interna de G_{3_b} con salida $y_b = 1$. Estados: $s_a = 2$, $s_b = 2$, $u_0 = 1$, $u_1 = 1$, $\sigma = 0$.
- $t = 1$: Transición interna de S_1 con salida $y_s = 2$. Estados: $s_a = 2$, $s_b = 2$, $u_0 = 1$, $u_1 = 1$, $\sigma = \infty$.
- $t = 2$: Transición interna de G_{3_a} con salida $y_a = 2$. Estados: $s_a = 1$, $s_b = 2$, $u_0 = 2$, $u_1 = 1$, $\sigma = 0$.
- $t = 2$: Transición interna de G_{3_b} con salida $y_b = 2$. Estados: $s_a = 1$, $s_b = 1$, $u_0 = 2$, $u_1 = 2$, $\sigma = 0$.
- $t = 2$: Transición interna de S_1 con salida $y_s = 4$. ... etc...

Eventos Simultaneos

Bajo los mismos supuestos que antes, podríamos también obtener la siguiente secuencia:

- $t = 1$: Transición interna de G_{3_a} con salida $y_a = 1$. Estados: $s_a = 2$, $s_b = 1$, $u_0 = 1$, $u_1 = 0$, $\sigma = 0$.
- $t = 1$: Transición interna de S_1 con salida $y_s = 1$. Estados: $s_a = 2$, $s_b = 1$, $u_0 = 1$, $u_1 = 0$, $\sigma = \infty$.
- $t = 2$: Transición interna de G_{3_b} con salida $y_b = 1$. Estados: $s_a = 2$, $s_b = 2$, $u_0 = 1$, $u_1 = 1$, $\sigma = 0$.
- $t = 1$: Transición interna de S_1 con salida $y_s = 2$. Estados: $s_a = 2$, $s_b = 2$, $u_0 = 1$, $u_1 = 1$, $\sigma = \infty$.
- $t = 2$: Transición interna de G_{3_a} con salida $y_a = 2$. Estados: $s_a = 1$, $s_b = 2$, $u_0 = 2$, $u_1 = 1$, $\sigma = 0$.
- $t = 2$: Transición interna de S_1 con salida $y_s = 3$. ... etc...

Eventos Simultáneos. Función de Desempate

Para evitar estas incongruencias, la especificación de acoplamiento cuenta con la función de desempate *Select*, que establece prioridades entre componentes que agendan una transición interna para el mismo instante de tiempo.

Dado un modelo acoplado $N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select)$, la función $Select : 2^D \to D$ establece para cada subconjunto de D quien es el modelo d que tiene mayor prioridad.

Dado que puede ser engorroso definir la función *Select* para cada subconjunto de D, en la práctica se suele establecer una lista de orden de prioridades. En nuestro ejemplo, si la lista de prioridades es G_{3_a}, G_{3_b}, S_1 obtenemos el resultado de la primer *simulación*. Si en cambio es S_1, G_{3_a}, G_{3_b} obtenemos el resultado de la segunda.

Acoplamiento y Legitimidad

Consideremos el siguiente modelo DEVS atómico:

$$M_3 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta) \text{ con:}$$
$$X = Y = S = \mathbb{R}^+$$
$$\delta_{\text{int}}(s) = \infty; \quad \delta_{\text{ext}}(s, e, x) = x;$$
$$\lambda(s) = s/2; \quad ta(s) = s;$$

y supongamos que acoplamos dos modelos M_3 de forma que la entrada de cada uno de ellos es la salida del otro.

Pese a que los dos modelos son legítimos, el acoplamiento no lo es. Evidentemente, la legitimidad no es cerrada frente al acoplamiento.

Parallel DEVS

En el formalismo DEVS, los conflictos de simultaneidad se resuelven mediante la función de desempate *Select*. Parallel DEVS brinda una alternativa que permite la ocurrencia simultanea de eventos:

- En los modelos atómicos se agrega una función de transición confluente δ_{con} que define el nuevo estado cuando al mismo tiempo se reciben eventos de entrada y se realiza una transición interna.
- Los eventos de entrada recolectan los valores de todos los eventos de salida que hayan ocurrido y que deban propagarse hacia el modelo en cuestión.
- Tanto la función δ_{con} como la δ_{int} dependen del conjunto de eventos recibidos.

Cell–DEVS

Cell–DEVS es una extensión para construir autómatas celulares basados en DEVS:

- Se define una estructura celular, en la cual cada componente atómico tiene cierto número de vecinos.
- Los eventos se transmiten entre vecinos de acuerdo a la estructura.
- El modelo acoplado de esta forma puede a su vez acoplarse con otros modelos DEVS.
- El formalismo incorpora algunos elementos a nivel atómico que son útiles en el contexto de autómatas celulares.
- Es muy conveniente para modelar y simular sistemas con muchos componentes conectados de manera regular.

Vectorial DEVS

Vectorial DEVS es una extensión que permite incluir arreglos de modelos DEVS:

- VecDEVS permite armar modelos de gran escala con estructuras repetitivas de manera simple.
- Además, el formalismo permite realizar particionado automático de modelos para la simulación en paralelo.
- En PowerDEVS hay implementada una librería de modelos VecDEVS.

Stochastic DEVS

STDEVS es una extensión para modelar sistemas estocásticos basado en DEVS:

- Formalmente, los modelos atómicos reemplazan las funciones de transición (externa e interna) por espacios de probabilidades.
- Los modelos atómicos STDEVS pueden acoplarse normalmente con otros modelos STDEVS o DEVS.
- Está demostrado que DEVS es un caso particular de STDEVS.
- En la práctica, los modelos STDEVS se simulan incorporando funciones RND en las transiciones.

Organización de la Presentación

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.

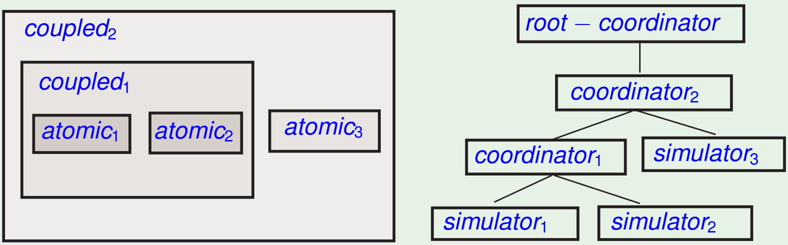
Idea Básica del Algoritmo de Simulación DEVS

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
    $s = \delta_{\rm int}(s)$
    $tl = t$
    $tn = t + ta(s)$
  when receive x-message (x, t) at time t
    $e = t - tl$
    $s = \delta_{\rm ext}(s, e, x)$
    $tl = t$
    $tn = t + ta(s)$
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

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:

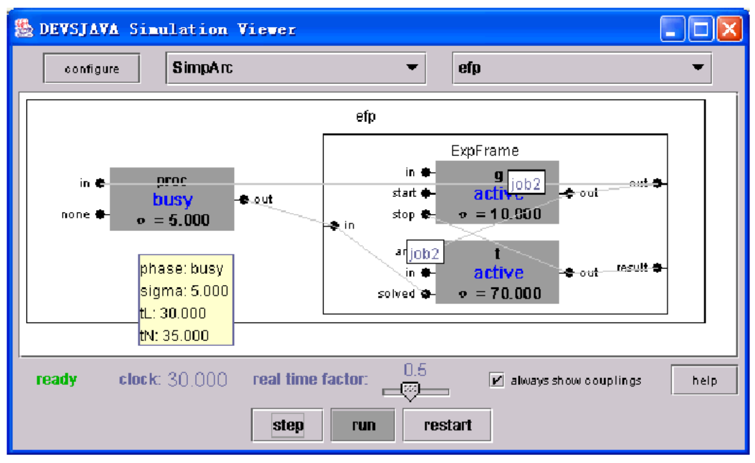
- ADEVS, DEVS/C++ y DEVSJAVA (University of Arizona).
- DEVSSim++ (KAIST, Corea).
- CD++ (Carleton University y UBA).
- LSIS-DME (LSIS, Marsella, Francia)
- VLE (Université du Littoral Côte d'Opale, Francia).
- SmallDEVS (Brno University of Technology, República Checa).
- JDEVS (Université de Corse).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

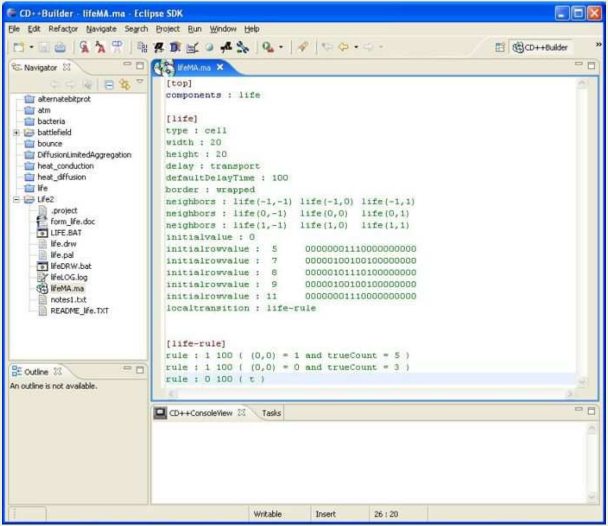

DEVSJAVA Simulation Viewer
configure
SimpArc
efp
efp
ExpFrame
proc
busy
σ = 5.000
in
none
out
in
g
job2
active
σ = 10.000
in
start
stop
out
out
job2
t
active
σ = 70.000
in
solved
out
result
phase: busy
sigma: 5.000
tL: 30.000
tN: 35.000
ready
clock: 30.000
real time factor:
0.5
always show couplings
help
step
run
restart

CD++

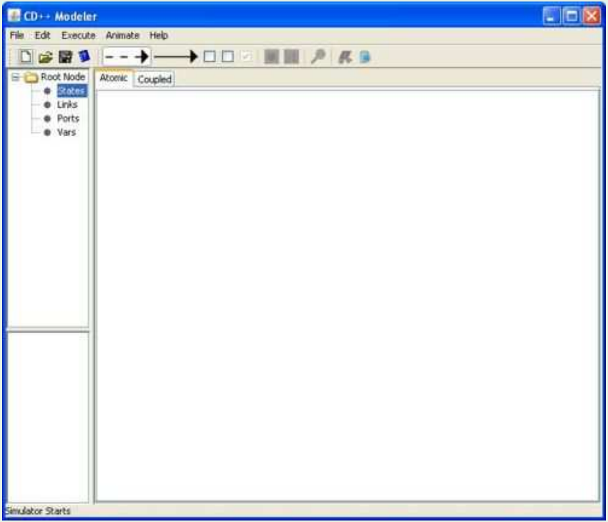
CD++ es una herramienta desarrollada por el grupo de Gabriel Wainer (Carleton University y Universidad de Buenos Aires), orientada a la implementación de Cell–DEVS.

- 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.

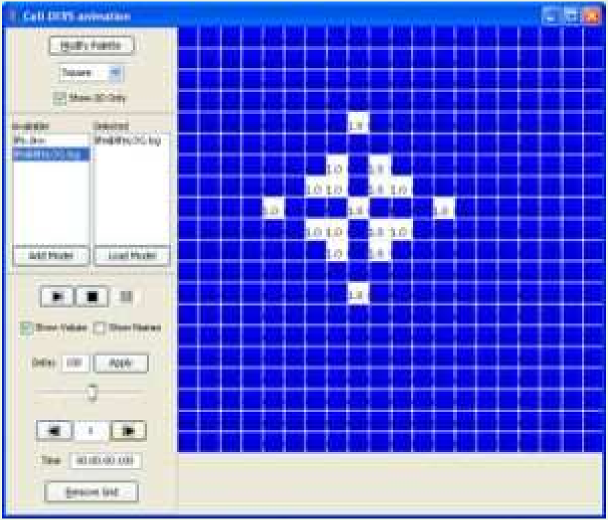
CD++


CD++Builder - lifeMA.ma - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
CD++Builder
Navigator
alternatebitprot
atm
bacteria
battlefield
bounce
DiffusionLimitedAggregation
heat_conduction
heat_diffusion
life
Life2
.project
form_life.doc
LIFE.BAT
life.drw
life.pal
lifeDRW.bat
lifeLOG.log
lifeMA.ma
notes1.txt
README_life.TXT
Outline
An outline is not available.
lifeMA.ma
[top]
components : life
[life]
type : cell
width : 20
height : 20
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1) life(0,0) life(0,1)
neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialrowvalue : 5 00000001110000000000
initialrowvalue : 7 00000100100100000000
initialrowvalue : 8 00000101110100000000
initialrowvalue : 9 00000100100100000000
initialrowvalue : 11 00000001110000000000
localtransition : life-rule
[life-rule]
rule : 1 100 { (0,0) = 1 and trueCount = 5 }
rule : 1 100 { (0,0) = 0 and trueCount = 3 }
rule : 0 100 { t }
CD++ConsoleView
Tasks
Writable
Insert
26 : 20

CD++


CD++ Modeler
File Edit Execute Animate Help
Root Node
States
Links
Ports
Vars
Atomic
Coupled
Simulator Starts

CD++

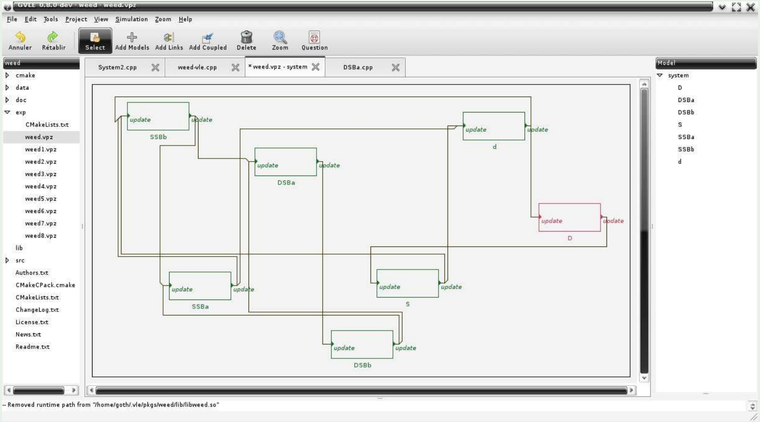


VLE

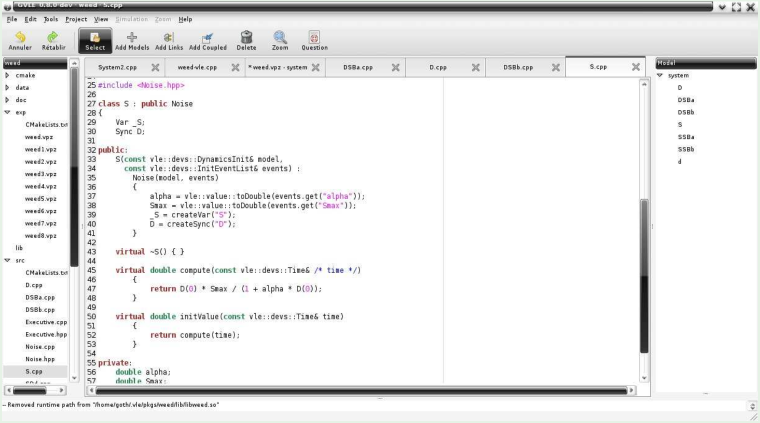
VLE (Virtual Laboratory Environment) es una herramienta nueva desarrollada por el grupo de Eric Ramat (Université du Littoral Côte d'Opale).

- 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


GVLE 0.8.0-dev - weed - weed.vpz
File Edit Tools Project View Simulation Zoom Help
Annuler Rétablir Select Add Models Add Links Add Coupled Delete Zoom Question
weed
cmake
data
doc
exp
CMakeLists.txt
weed.vpz
weed1.vpz
weed2.vpz
weed3.vpz
weed4.vpz
weed5.vpz
weed6.vpz
weed7.vpz
weed8.vpz
lib
src
Authors.txt
CMakeCPack.cmake
CMakeLists.txt
ChangeLog.txt
License.txt
News.txt
Readme.txt
System2.cpp
weed-vle.cpp
* weed.vpz - system
DSBa.cpp
SSBb
DSBa
d
D
SSBa
S
DSBb
update
Model
system
D
DSBa
DSBb
S
SSBa
SSBb
d
-- Removed runtime path from "/home/goth/.vle/pkgs/weed/lib/libweed.so"

VLE

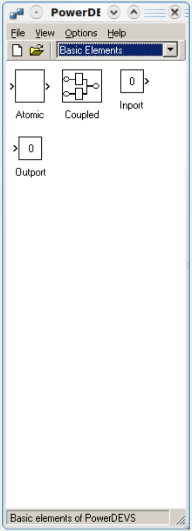

GVLE 0.8.0-dev - weed - S.cpp
File Edit Tools Project View Simulation Zoom Help
Annuler Rétablir Select Add Models Add Links Add Coupled Delete Zoom Question
System2.cpp weed-vle.cpp *weed.vpz - system DSBa.cpp D.cpp DSBb.cpp S.cpp
weed
cmake
data
doc
exp
CMakeLists.txt
weed.vpz
weed1.vpz
weed2.vpz
weed3.vpz
weed4.vpz
weed5.vpz
weed6.vpz
weed7.vpz
weed8.vpz
lib
src
D.cpp
DSBa.cpp
DSBb.cpp
Executive.cpp
Executive.hpp
Noise.cpp
Noise.hpp
S.cpp
#include <Noise.hpp>
class S : public Noise
{
Var _S;
Sync D;
public:
S(const vle::devs::DynamicsInit& model,
const vle::devs::InitEventList& events) :
Noise(model, events)
{
alpha = vle::value::toDouble(events.get("alpha"));
Smax = vle::value::toDouble(events.get("Smax"));
_S = createVar("S");
D = createSync("D");
}
virtual ~S() { }
virtual double compute(const vle::devs::Time& /* time */)
{
return D(0) * Smax / (1 + alpha * D(0));
}
virtual double initValue(const vle::devs::Time& time)
{
return compute(time);
}
private:
double alpha;
double Smax;
Model
system
D
DSBa
DSBb
S
SSBa
SSBb
d
-- Removed runtime path from "/home/goth/.vle/pkgs/weed/lib/libweed.so"

PowerDEVS

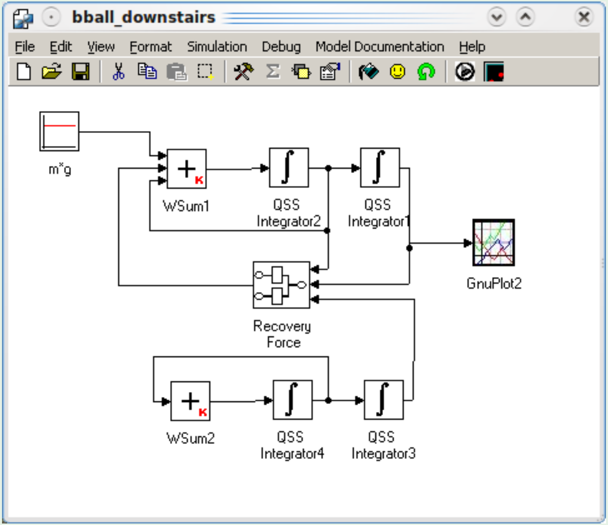
PowerDEVS es una herramienta desarrollada en la Universidad Nacional de Rosario, con las siguientes características:

- 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.

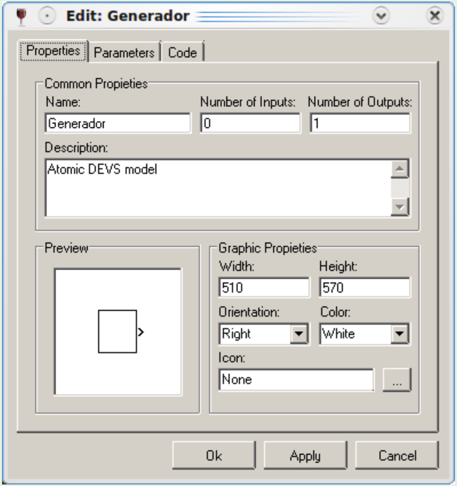
PowerDEVS. Ventana de librerías



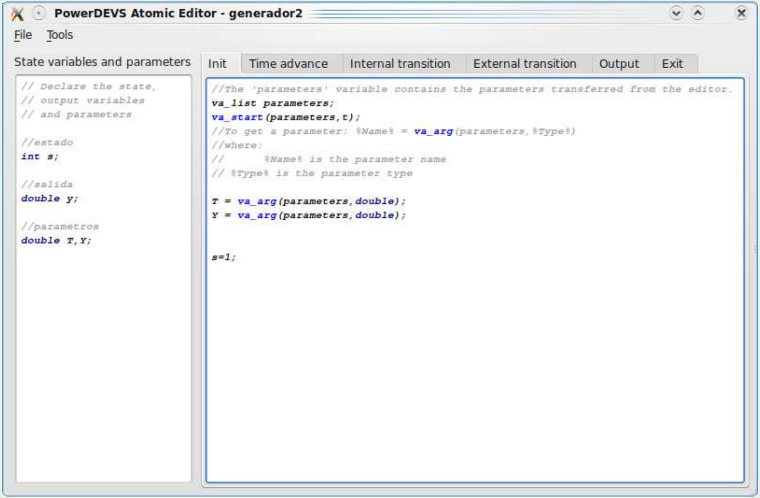
PowerDEVS. Ventana de modelo



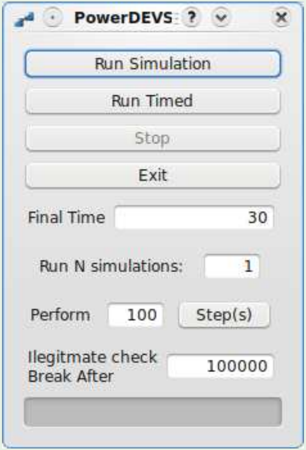
PowerDEVS. Ventana de edición de bloques



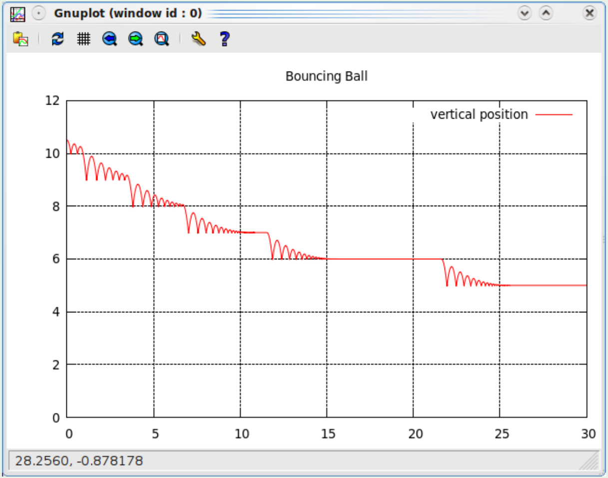
PowerDEVS. Editor de modelos atómicos



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:

$$P_2 = < X, Y, S, \delta_{\mathrm{int}}, \delta_{\mathrm{ext}}, \lambda, ta >$$

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

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

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

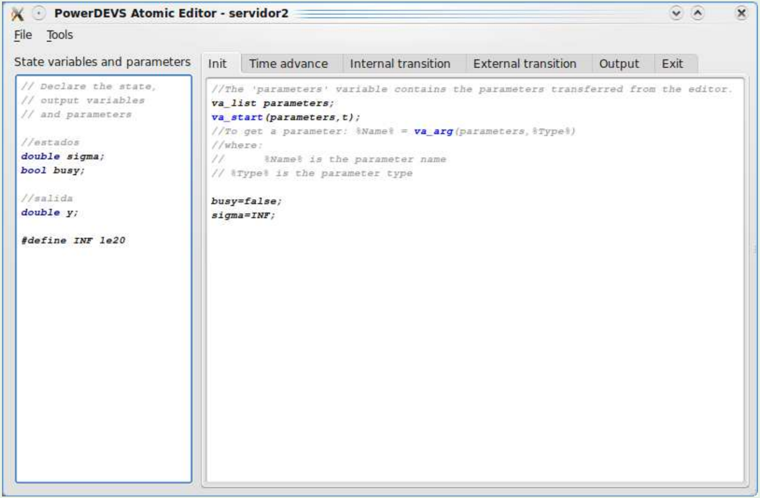
$$\delta_{\mathrm{int}}(s) = (\infty, false)$$

$$\lambda(s) = 1$$

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

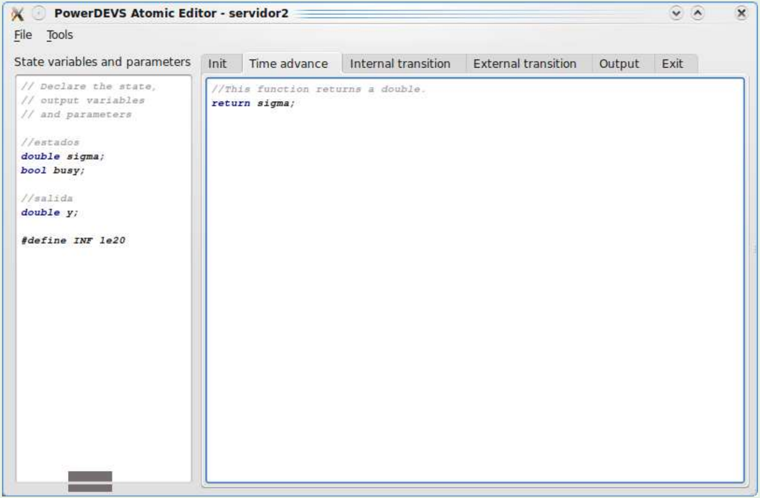
PowerDEVS. Ejemplo Modelo Atómico

Declaraciones y Estado inicial



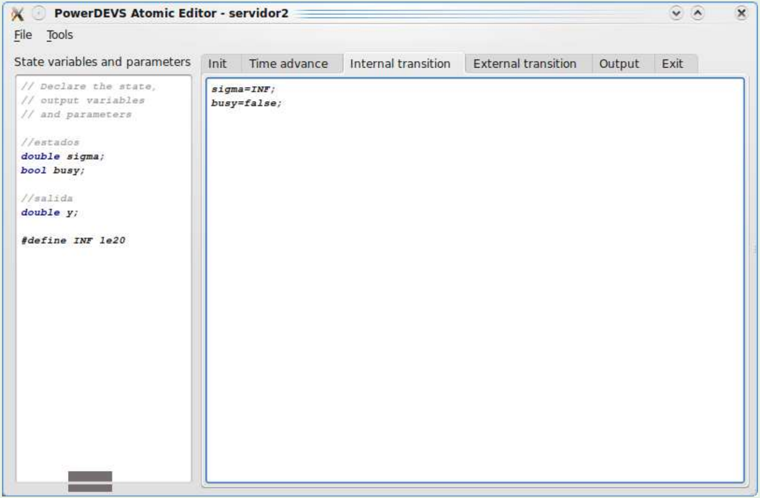
PowerDEVS. Ejemplo Modelo Atómico

Función de avance de tiempo



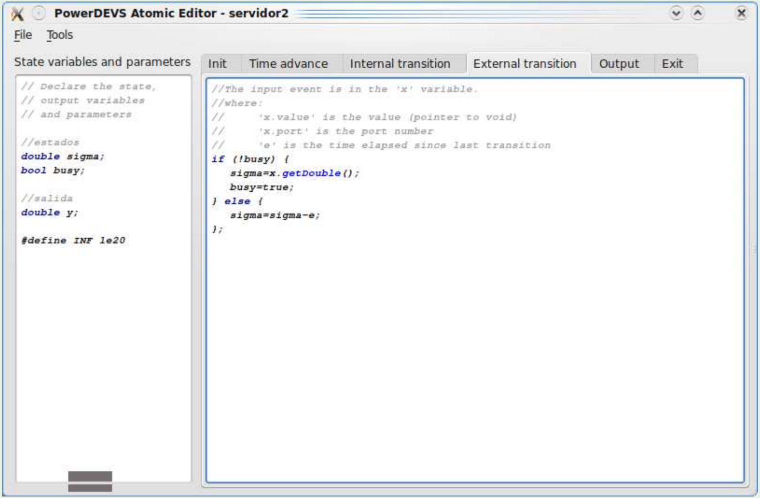
PowerDEVS. Ejemplo Modelo Atómico

Función de transición interna



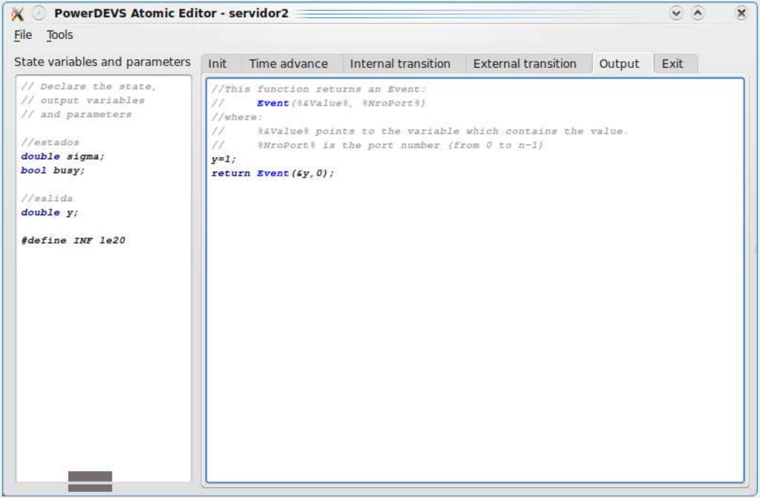
PowerDEVS. Ejemplo Modelo Atómico

Función de transición externa



PowerDEVS. Ejemplo Modelo Atómico

Función de salida



Bibliografía

- Christos Cassandras and Stéphane Lafortune.
  Introduction to Discrete Event Systems. Second Edition.
  Springer, New York, 2008.
- Ernesto Kofman.
  Introducción a la Modelización y Simulación de Sistemas de Eventos Discretos con el Formalismo DEVS.
  Facultad de Cs. Exactas, Ing. y Agrimensura. UNR, 2003.
- B.P. Zeigler, A. Muzy, and E. Kofman.
  Theory of Modeling and Simulation. 3rd Edition.
  Academic Press, New York, 2018.