Formalismo DEVS - Teoría y Aplicaciones

Ernesto Kofman

1. Introducción.

Durante las últimas décadas, la rápida evolución de la tecnología ha producido una proliferación de nuevos sistemas dinámicos, generalmente hechos por el hombre y de gran complejidad. Ejemplos de ellos son las redes de computadoras, sistemas de producción automatizados, de control de tráfico aéreo; y sistemas en general de comando, de control, de comunicaciones y de información. Todas las actividades en estos sistemas se deben a la ocurrencia asincrónica de *eventos discretos*, algunos controlados (tales como el pulsado de una tecla) y otros no (como la falla espontánea de un equipo). Esta característica es la lleva a definir el término de *Sistemas de Eventos Discretos*.

Las herramientas matemáticas que hoy disponemos (básicamente ecuaciones diferenciales y en diferencias) fueron desarrolladas durante los últimos doscientos años para modelar y analizar los procesos *conducidos por el tiempo* que generalmente uno encuentra en la naturaleza. El proceso de adaptar estas herramientas y desarrollar nuevas para los sistemas *conducidos por eventos* tiene solo unos pocos años. (Cassandras, 1993). Por este motivo, encontramos en la teoría de los sistemas de eventos discretos no sólo una serie de herramienta específicas para atacar problemas de modelización, simulación y análisis de sistemas altamente ligados a la práctica de la ingeniería y a los problemas de la informática, sino también un campo fértil para el desarrollo de nuevas técnicas y teorías debido a la cantidad de problemas aún abiertos en el área.

Dentro de los formalismos mas populares de representación de sistemas de eventos discretos (DES) están las *Redes de Petri*, las *Statecharts*, *Grafcet*, *Grafos de Eventos* y muchas generalizaciones y particularizaciones de los mismos. Con respecto a las herramientas de análisis, sin dudas las más interesantes son las obtenidas con la introducción de estructuras algebraicas de tipo dioides *max-plus* y *min-plus* (Baccelli et al. 1992). Nos ocuparemos, sin embargo, exclusivamente de modelización y simulación, dejando de lado estas herramientas de análisis.

Orientado a los problemas de modelización y simulación de DES, en la década del 70, el matemático Bernard Zeigler propuso un formalismo general para la representación de dichos sistemas. Este formalismo, denominado DEVS (Discrete EVent System specification), es de hecho el formalismo más general para el tratamiento de DES. El hecho de estar fundado en la base de la teoría de sistemas, lo convierte en un formalismo universal, y por lo tanto, todos los otros formalismos mencionados en el párrafo anterior pueden ser *absorbidos* por DEVS (es decir, todos los modelos representables en dichos formalismos pueden ser representados en DEVS). (Zeigler and Vahie, 1993).

Mas aún, problemas más generales que la modelización y simulación, como ser análisis y diseño, no solo ya en DES, sino también en Sistemas Híbridos, pueden ser abordados a partir del formalismo DEVS. (Zeigler et al., 1995).

Veremos entonces varios de los principales aspectos del formalismo DEVS, basándonos principalmente en el libro *Teoría de Modelización y Simulación* (Zeigler et al., 2000) y salvo que se cite a otra fuente, estaremos en general refiriéndonos a dicho libro. Además de algunas aplicaciones directas, estudiaremos en particular uno de los mas recientes desarrollos consistente en la utilización del formalismo DEVS para la simulación de Sistemas Continuos.

2. Definiciones Previas.

Como dijimos antes, la definición del formalismo DEVS se basa en la teoría de sistemas. Antes de abordar la definición de DEVS, daremos una recorrida rápida por algunos conceptos fundamentales en la teoría de sistemas que aplicaremos mas adelante en la definición de dicho formalismo.

2.1. Clasificación de Sistemas

La Teoría de Sistemas clasifica los sistemas, básicamente, según dos grandes aspectos:

- **Nivel de especificación del sistema:** Se definen diferentes niveles en los cuales es posible describir el comportamiento del sistema y los mecanismos que producen ese comportamiento.

- **Formalismo de especificación del sistema:** Diferentes formas de modelado que conducen a especificaciones continuas o discretas, ya sean en el tiempo o en las variables descriptivas.

Con respecto al nivel de especificación de un sistema, básicamente se hace una distinción entre la *estructura* (constitución interna) y el *comportamiento* del sistema (manifestación externa del mismo). El *comportamiento* del sistema es la relación entre la historia temporal de entrada y la historia temporal de salida. La *estructura* interna en cambio nos da una visión de cómo se conectan internamente los diferentes componentes del sistema. Conocer la *estructura* de un sistema permite deducir su *comportamiento*, mientras que en general, la recíproca no conduce a una única solución. En base a esto, la especificación de un modelo podrá hacerse dentro una de las dos clases. Una especificación de comportamiento nos dará una visión de caja negra del mismo, mientras que una especificación estructural nos dará mucho mas información sobre el sistema en cuestión.

Una clasificación quizás mas completa de los modelos según su nivel de descripión es la que da Zeigler bajo el título de *Especificación Jerárquica de Sistemas*. Esta clasificación se basa en consideraciones de dinámica y de modularidad de los sistemas. La siguiente tabla muestra un resumen de la misma:

Nivel	Nombre	Conocimiento
0	Marco de Observación	Cuales son las entradas a considerar, que variables medir y como observarlas sobre una base de tiempo.
1	Comportamiento Entrada/Salida	Datos colectados e indexados en el tiempo desde el sistema fuente, consistentes en pares de Entrada/Salida
2	Función Entrada/Salida	Conocimiento del estado inicial y a partir del mismo, la relación única existente entre las trayectorias de entrada y salida.
3	Transición de Estados	Como los estados son afectados por las entradas y como las salidas son afectados por la entrada y el estado actual.
4	Componentes acoplados	Que componentes tiene el sistema y como se acoplan entre si. Los componentes pueden estar dados a bajo nivel o pueden ser nuevas estructuras.

Tabla 1: *Niveles de especificación jerárquica*

Como puede verse, a medida que subimos de nivel, vamos teniendo cada vez mas información sobre el sistema.

La forma de movernos dentro de este cuadro dependerá del tipo de problema. En la teoría de sistemas podemos distinguir tres problemas básicos:

Análisis de Sistemas: Consiste en predecir el comportamiento basándose en el conocimiento de la estructura del sistema. Implica ir desde los niveles mas altos hacia los mas bajos de conocimiento.

Deducción de Sistemas: Se trata de deducir como es la estructura basándose en el conocimiento del comportamiento del sistema. Implica, en este caso, moverse desde los niveles bajos hacia los mas altos.

Diseño de Sistemas: En este caso, el sistema aún no existe, pero se cuenta con una especificación de su comportamiento. Implica también ir desde los niveles bajos hacia los altos.

En cuanto a los formalismos de especificación de sistemas, encontramos básicamente tres grandes clases, clasificadas de acuerdo a la forma de cambio de las variables. Los sistemas en los que las variables varían continuamente (en una base de tiempo continua) se denominan simplemente Sistemas Continuos. Cuando las variables están definidas en una base de tiempo discreta, se habla de Sistemas de Tiempo Discreto. Por último, los sistemas que nos ocuparán (Sistemas de Eventos Discretos) son aquellos cuyas variables evolucionan de manera discreta en una base de tiempo en general continua. Sobre la base de estos formalismos, se definen también formalismos resultado de combinaciones de especificaciones diferentes, dando lugar a los denominados Sistemas Híbridos.

2.2. Especificación de Comportamiento de un Sistema Dinámico

Veremos ahora una estructura que es capaz de representar el comportamiento de cualquier sistema dinámico determinista, independientemente de su característica continua, discreta o híbrida.

Formalmente, una especificación de *comportamiento* se define como una estructura del tipo:

$S = \langle T, X, Y, \Omega, Q, \Delta, \Lambda \rangle$, donde:

T : base de tiempo.
X : conjunto de valores de entrada.
Y : conjunto de valores de salida.
Ω : conjunto de los segmentos de entrada admisibles; $\omega: \langle t_1, t_2 \rangle \rightarrow X$ sobre T.
Q: conjunto de valores de los estados.
Δ: Q x $\Omega \rightarrow Q$ función de transición global.
Λ: Q x $X \rightarrow Y$ función de salida.

La interpretación puede ser la siguiente: dado un segmento de trayectoria de entrada (dentro del conjunto de los segmentos de entrada admisibles Ω) y un valor inicial del estado (dentro del conjunto de valores de los estados Q) correspondiente al instante inicial de dicho segmento, la función de transición utiliza estos elementos como argumento y calcula el nuevo valor del estado, correspondiente al instante de tiempo del final del segmento de entrada mencionado. La salida del sistema, en tanto, es simplemente un reflejo del estado (a través de la función de salida por supuesto).

3. Bases del Formalismo DEVS.

Definiremos a continuación el formalismo DEVS para modelos atómicos (no acoplados, es decir, modelos de *comportamiento*) y estudiaremos su relación con la definición general de un sistema dinámico.

3.1. Segmentos de Eventos.

Antes de entrar en la definición formal de DEVS, definiremos que es un *segmentos de eventos*. La Figura 1 muestra la forma de este tipo de trayectorias.

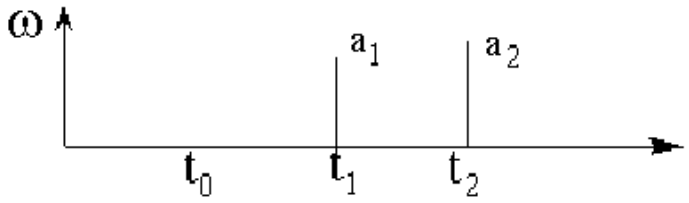


Figura 1. Un segmento de Eventos

Un segmento de eventos se define formalmente de la siguiente forma:
Sea $\omega:(t_0, t_n) \rightarrow A \cup \{\varnothing\}$ un segmento sobre una base de tiempo continua (o sea, una función de t perteneciente al intervalo (t_0, t_n) de los reales) y el conjunto $A \cup \{\varnothing\}$. Aquí $\varnothing$ es un elemento que no pertenece a A y representa la ausencia de evento. Luego ω es un segmento de eventos si existe un conjunto finito de puntos $t_1, t_2, t_3, ..., t_{n-1} \in (t_0, t_n)$ tales que $\omega(t_i) = a_i \in A$ para $i = 1, ..., n\text{-}1$, y $\omega(t) = \varnothing$ para todo otro $t \in (t_0, t_n)$.

3.2. Definición del Formalismo DEVS.

Un modelo DEVS atómico se define como una estructura:

$M = \langle X, S, Y, d_{\text{int}}, d_{ext}, l, ta \rangle$ donde:

X : conjunto de valores de las entradas

S : conjunto de valores de los estados

Y : conjunto de valores de las salidas

d_{int} : función de transición interna: $d_{\text{int}} : S \rightarrow S$

d_{ext} : función de transición externa: $d_{\text{ext}} : Q \text{ x } X \rightarrow S$, donde
$Q = \{(s, e) \mid s \in S, \ 0 \le e \le ta(s)\}$ es el conjunto de estado total
e es el tiempo transcurrido desde la última transición

λ : función de salida: $\lambda : S \rightarrow Y$

ta : es la función de avance de tiempo: $ta : S \rightarrow \Re_0^+ \infty$

La interpretación de estos elementos es la siguiente: En algún instante de tiempo el sistema llega al estado *s*. Si no ocurre ningún evento externo el sistema permanecerá en dicho estado *s* durante un tiempo *ta*(*s*). Cuando el tiempo transcurrido, *e*, iguala a *ta*(*s*), el sistema produce un evento de salida con valor $\lambda(s)$ y cambia al estado s'=$\delta_{int}(s)$. Se dice entonces que el sistema efectuó una transición interna.

Si antes que ocurra la transición interna ocurriese un evento externo con valor *x*, o sea, que dicho evento ocurre mientras el sistema está en el estado total (*s*,*e*), siendo e ≤ *ta*(*s*), el sistema cambia al estado $\delta_{ext}(s,e,x)$, pero en este caso no se produce ningún evento de salida.

Como puede verse en la definición, *ta*(*s*) puede ser en general un número real, inclusive 0 ó ∞. Si este tiempo (denominado también *tiempo de vida del estado*) es 0, diremos que *s* es un estado *transitorio*. Si en cambio, *ta*(*s*) es ∞, el sistema estará en el estado *s* para siempre, a menos que ocurra algún evento externo que lo saque del mismo. En este caso diremos que el estado *s* es un estado *pasivo*.

De todo esto podemos deducir que las trayectorias de entrada de un DEVS serán segmentos de eventos que ocurrirán en un tiempo cualquiera. Las trayectorias de estados, en tanto, serán seccionalmente constantes, y las trayectorias de salida serán, al igual que las de entradas, segmentos de eventos. Las trayectorias del tiempo transcurrido, en cambio, serán del tipo *diente de sierra*. En la Figura 2 pueden verse las diferentes trayectorias mencionadas.

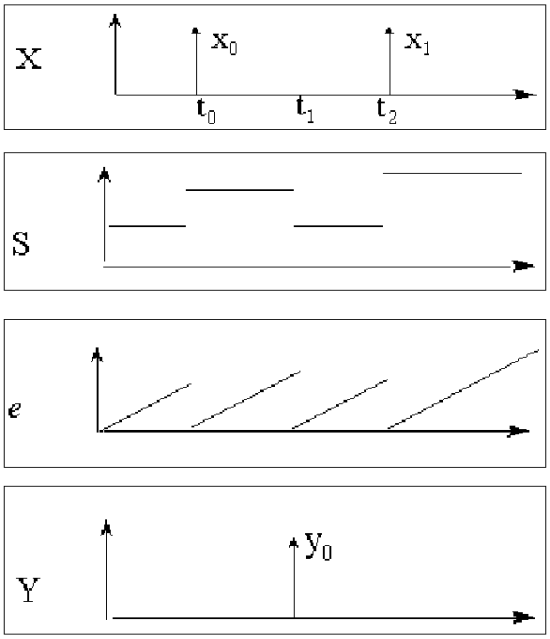


Figura 2. Trayectorias DEVS

Ejemplo 1: Un modelo simple del comportamiento de un procesador puede ser el siguiente: al sistema llegan eventos representando trabajos a realizar. Cada trabajo tiene asociado un tiempo (conocido) de procesamiento. Una vez transcurrido este tiempo, el procesador provoca un evento en el que muestra el "resultado" del trabajo (una función *y*(.) conocida del trabajo en cuestión). Si cuando llega un evento el procesador estaba procesando un trabajo, este evento debe ser ignorado. Como puede verse, no estamos teniendo en cuenta lo que internamente realiza el procesador. Solo consideramos el tiempo que lleva hacerlo (y lo suponemos conocido).

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

$X = \{job_1, job_2, ..., job_n\}$

$$S = \{job_1, job_2, ..., job_n\} \cup \{f\} \times \Re_0^+$$
$$Y = \{y(job_1), y(job_2), ..., y(job_n)\}$$
$$d_{int}(job, s) = (f, \infty)$$
$$d_{ext}(job, s, e, x) = \begin{cases} (x, t_p(x)) & si\ job = f \\ (job, s - e)\ en\ otro\ caso \end{cases}$$
$$l(job, s) = y(job)$$
$$ta(job, s) = s$$

Ejemplo 2: Considerar una cola FIFO, de capacidad de almacenamiento infinita (esto no es realista, pero simplifica el modelo), a la cual llegan trabajos para ser almacenados y eventualmente señales de "ready" indicando que debe transmitir (y descargar) el primer trabajo de dicha cola. La transmisión de dicho trabajo se realiza mediante un evento de salida con dicho trabajo. Suponiendo que la cola demora un tiempo nulo en provocar la salida, el modelo es el siguiente:

$M_Q = \langle X, S, Y, d_{\text{int}}, d_{ext}, l, ta \rangle$, donde
$$X = \{job_1, job_2, ..., job_n\} \cup \{"ready"\} = J \cup \{"ready"\}$$
$$S = J^+ \cup \{f\} \times \{0; \infty\}$$
$$Y = J$$
$$d_{int}(q \bullet job, s) = (q, \infty)$$
$$d_{ext}(q, s, e, x) = \begin{cases} (x \bullet q, \infty) & si\ x \in J \\ (q, 0) & en\ otro\ caso \end{cases}$$
$$l(q \bullet job, s) = job$$
$$ta(job, s) = s$$

Ejemplo 3: (ejercicio). Modificar el Ejemplo 1 de forma tal que cuando llegue un trabajo al procesador, este sea procesado aunque para esto haya que descartar el trabajo que se estaba procesando.

Ejemplo 4: (ejercicio). Modificar el Ejemplo 2 para representar un retardo entre la llegada de la señal de "ready" y la transmisión del trabajo correspondiente.

Ejemplo 5: (ejercicio). En el ejemplo anterior, realizar las modificaciones necesarias para que cuando llegue un evento de "ready" y la cola esté vacía, ante la próxima llegada de un trabajo la cola envíe inmediatamente este trabajo (y no se quede esperando una nueva señal de "ready").

Ejemplo 6: (ejercicio). Modificar el ejemplo anterior para limitar la capacidad de almacenamiento de la cola, de manera tal que cuando se reciba un nuevo trabajo y la cola esté llena (suponer una capacidad para *n* trabajos), este nuevo trabajo sea ignorado.

3.2.1. Conflictos de Simultaneidad.

Según la definición hecha del formalismo DEVS, no queda definido que es lo que ocurre cuando se produce un evento de entrada en el mismo instante que estaba prevista una transición interna. Las opciones son claramente dos: o bien se ejecuta primero la transición interna y luego la transición externa o bien se ejecuta la transición externa (y se ignora la transición interna que iba a producirse). Ambas alternativas son igualmente válidas y pueden adoptarse en función del comportamiento del sistema que se desea modelar.

Para solucionar esta indeterminación, se propuso una extensión del formalismo, denominada *Parallel-DEVS*, en la cual los eventos simultáneos son tratados por una nueva función de

transición denominada *función de transición confluente*. La denominación proviene del hecho que su principal aplicación es en la simulación de procesos "en paralelo".

3.3. Sistema Dinámico definido por DEVS.

En la sección 2.2 se vio una estructura capaz de representar cualquier sistema dinámico independiente de cual sea el paradigma de modelado. Es importante, desde el punto de vista formal, verificar que los modelos DEVS definan sistemas dinámicos (ya que de lo contrario, no funcionarán) y establecer, si fuera necesario, condiciones para que esto ocurra. Por otro lado, la obtención del sistema dinámico especificado por DEVS nos ayudará a comprender de una forma mucho más acabada la esencia de la semántica operacional de este formalismo.

La estructura a la que hacemos referencia en el párrafo anterior era:

$S = \langle T, X, Y, \Omega, Q, \Delta, \Lambda \rangle$.

Para que el sistema $DEVS = \langle X', S, Y', d_{\text{int}}, d_{ext}, l, ta \rangle$ sea realmente un Sistema Dinámico, esta estructura debería ser un caso particular de la anterior. Veremos entonces como se relacionan las mismas:

- La base de tiempo T de la estructura S es el conjunto de los números reales $\hat{A}$.
- El conjunto de valores de entrada X es $X'^{\varnothing} = X' \cup \{\varnothing\}$, o sea, el conjunto de valores de entrada del sistema DEVS junto al símbolo ∅ que denota la ausencia de evento.
- El conjunto de valores de salida Y es $Y'^{\varnothing} = Y' \cup \{\varnothing\}$
- El conjunto de estados Q es el conjunto de estado total Q' de DEVS.
- El conjunto Ω de segmentos de entrada admisible es el conjunto de los segmentos de eventos con valores en X.
- Las trayectorias de estado son seccionalmente constantes sobre S y T;
- Las trayectorias de salidas son segmentos de eventos sobre Y y T;
- La función de transición Δ queda definida como sigue:

  Sea w: $(t_i , t_f] \rightarrow X^{\varnothing}$ un segmento de eventos y sea el estado $q = (s, e)$ el estado del sistema en tiempo t_i. Entonces:

  $\Delta (q, \omega(t_i , t_f]) =$

(1) $(s, e + t_f - t_i))$

si $e + t_f - t_i < ta(s) \wedge \neg\exists\, t \in (t_i , t_f] / \omega(t) \neq \varnothing$

(no ocurren eventos externos ni internos)

(2) $\Delta ((\delta_{int}(s), 0), \omega[t_i + ta(s) - e, t_f])$

si $e + t_f - t_i = ta(s) \wedge \neg\exists\, t \in (t_i, t_i + ta(s) - e) / \omega(t) \neq \varnothing$

(ocurre primero un evento interno)

(3) $\Delta((d_{ext}((s, e+t-t_i), \omega(t)), 0), \varnothing_{[t, t\,]} \bullet \omega_{[t, t_f]})$

si $\exists\, t \in (t_i, \min\{t_f, t_i + ta(s)-e\}] : \omega(t) \neq \varnothing \wedge \neg\exists\, t' \;\hat{I}\; <t_i , t\,) : \omega(t') \neq \varnothing$

(ocurre primero un evento externo)

- La función de salida Λ está dada por:

$\Lambda(q, x)$	=	$\lambda(s)$	si $e = ta(s)$ y $\omega(t) = \varnothing$
	=	$\lambda(s)$ o $\varnothing$	si $e = ta(s)$ y $\omega(t) \neq \varnothing$
			(solo hay salida en eventos internos)
		$\varnothing$	en otro caso.

Queda además, considerar algunas restricciones para asegurar que la función Δ quede bien definida, esto es, asegurar que la definición recursiva lleve a un resultado. Una vez garantizado esto, podremos asegurar que la estructura especificada en el formalismo DEVS define realmente un sistema dinámico.

Las condiciones necesarias y suficientes para que la función recursiva mencionada quede bien definida serán tratadas en la siguiente sección

3.4. Legitimidad de un modelo DEVS.

Desde un punto de vista intuitivo puede verse que el único motivo por el cual un DEVS no definirá un sistema dinámico es cuando en un tiempo finito ocurra un número infinito de eventos internos, ya que en este caso las trayectorias de salida dejarán de ser segmentos de eventos. Este caso, visto desde la función de transición global del sistema dinámico que se intenta definir, implicará que la definición recursiva de la misma en algunos casos no llegue nunca al caso "raíz".

Para poder dar condiciones que nos aseguren que esto no ocurra, comenzaremos definiendo una función de transición interna extendida:

$$\delta_{int}^{+} : S \times \mathbf{I_0}^{+} \rightarrow S$$

definida recursivamente según:

$$\delta_{int}^{+}(s,0) = s$$
$$\delta_{int}^{+}(s,n+1) = \delta_{int}(\delta_{int}^{+}(s,n))$$

A partir de esta, definimos una nueva función:

$$\Sigma: S \times \mathbf{I_0}^{+} \rightarrow \mathbf{R_0}^{+}$$

tal que $\Sigma(s,0) = 0$

$$\Sigma(s,n) = \sum_{i=0}^{n-1} ta(d_{int}^{+}(s,i)).$$

Se puede demostrar entonces que la estructura especificada por un DEVS es un Sistema Dinámico sí y solo si:

$\lim_{n\to\infty} \Sigma(s,n) \rightarrow \infty$ para todo valor de s

En este caso, se dice que el sistema DEVS es *legitimado*. La interpretación es sencilla: la función $\Sigma(s,n)$ nos da el tiempo que se demora en realizar n transiciones internas partiendo del estado inicial s (suponiendo que no hay eventos externos en todo este tiempo). Por lo tanto, si cuando el número de eventos tiende a infinito esta función también tiende a infinito, no podrá jamás haber un número de eventos infinito en tiempo finito.

Cuando existe un ciclo cerrado en el diagrama de estados de d_{int} (o sea, una sucesión cíclica de estados producto de transiciones internas sucesivas) que contiene solamente estados transitorios el DEVS es no legitimado. Cuando el conjunto de estados S es finito, la no existencia de estos ciclos es condición suficiente para asegurar legitimidad. Sin embargo, en sistemas con infinitos posibles estados esta es solamente una condición necesaria (como ejemplo, uno se puede imaginar una sucesión de estados donde el tiempo de vida de un estado es la mitad del tiempo de vida del estado anterior, como en el ejemplo de Zenón sobre "Aquiles y la tortuga").

En casos mas generales, una condición suficiente para asegurar legitimidad es la existencia de un límite inferior positivo para la función $ta(s)$, aunque esta es una condición muy fuerte ya que nos prohibe tener estados transitorios. Una condición suficiente mas relajada es tener al estado dividido en dos conjuntos, uno finito en el cual la única restricción es que no haya ciclos cerrados de tiempo de vida nulo y otro conjunto (posiblemente infinito) en el cual haya una cota inferior positiva para el tiempo de vida de los estados.

Ejemplo 7: (ejercicio). Verificar la legitimidad de todos los modelos obtenidos en los ejemplos y ejercicios vistos hasta el momento.

Ejemplo 8: "Cinta transportadora". Una cinta transportadora, que forma parte de un sistema de producción más complejo, está comandada por una señal de control que le indica si debe marchar o no. La velocidad a la que avanza la cinta es constante e igual a v_c cuando está en funcionamiento.

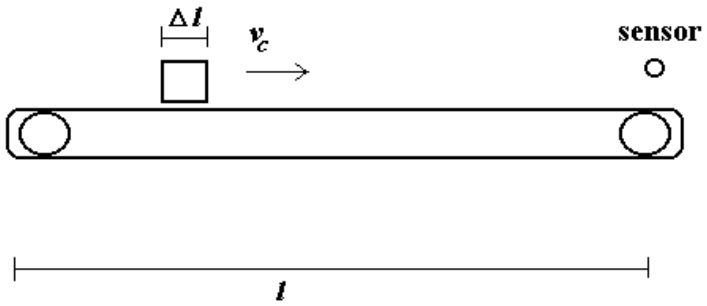


Figura 3. Cinta transportadora.

Provenientes de otra parte del sistema de producción, la cinta recibe piezas (de manera asincrónica). Cada vez que una pieza alcanza la posición de un sensor (ubicado a una distancia l del comienzo de la cinta), se debe provocar un evento de salida (naturalmente, este evento de salida deberá ser utilizado por el sistema de control). Luego de esto, cuando la pieza llega al final de la cinta (a una distancia $l+\Delta l$ del comienzo de la misma), la pieza sale de la cinta transportadora, provocando un nuevo evento de salida (que indica el paso de la pieza a otro sector del sistema y el evento producido por el sensor cuando deja de detectar la pieza).

Para realizar el modelo, supondremos como simplificación que el diseño del sistema es tal que no puede haber simultáneamente mas de una pieza en la cinta. Con esta hipótesis, proponemos el siguiente modelo:

$M_C = \langle X, S, Y, d_{int}, d_{ext}, l, ta \rangle$, donde

$X = \{start, stop, arrive\}$
$S = \Re \times \{0, v_c\} \times \Re_0^+$
$Y = \{detect, leave\}$

$$d_{int}(d, v, s) = \begin{cases} (-\infty, v, \infty) & si\ d \geq l \\ (l, v, \frac{\Delta l}{v_c}) & en\ otro\ caso \end{cases}$$

$$d_{ext}(d, v, s, e, x) = \begin{cases} (0, v, \frac{l}{v}) & si\ x = arrive \\ (d', 0, \infty) & si\ x = stop \\ (d', v_c, \frac{\Delta}{v_c}) & en\ otro\ caso \end{cases}$$, donde $d' = d + e \cdot v$ $\Delta = \begin{cases} l - d'\ si\ d < l \\ l + \Delta l - d'\ \ en\ otro\ caso \end{cases}$

$$l(d, v, s) = \begin{cases} leave & si\ d \geq l \\ detect & en\ otro\ caso \end{cases}$$
$ta(d, v, s) = s$

Ejemplo 9: En el ejemplo anterior, realizaremos modificaciones para admitir la posibilidad de que haya varias piezas simultáneamente en la cinta. Supondremos, de todas formas, que no pueden llegar piezas mientras haya una pieza al comienzo de la cinta.

Con todo esto, un modelo posible es el siguiente:

$M_C = \langle X, S, Y, d_{int}, d_{ext}, l, ta \rangle$, donde

$X = \{move, stop, arrive\}$
$S = \Re^* \times \{0, v_c\} \times \Re_0^+$, donde $\Re^*$ es el conjunto de las secuencias finitas de números reales incluyendo la secuencia vacía.
$Y = \{detect, leave\}$

Los elementos de la secuencia que forma parte del estado son las distancias recorridas por cada pieza.

$$d_{int}(q \bullet d_2 \bullet d_1, v, s) = \begin{cases} ((q \bullet d_2 \bullet d_1) + s \cdot v, v, \frac{\Delta l}{v}) & si\ d_1 < l \\ ((q \bullet d_2) + s \cdot v, v, \frac{l - d_2}{v}) & si\ d_1 \geq l \wedge d_2 \neq f \\ (f, v, \infty) & en\ otro\ caso \end{cases}$$

(el término $+s \cdot v$ se suma a todos los elementos de la secuencia precedente al mismo).

$$d_{ext}(q \bullet d_1, v, s, e, x) = \begin{cases} ((q \bullet d_1) + e \cdot v, 0, \infty) & si\ x = stop \\ ((q \bullet d_1) + e \cdot v, v_c, \frac{l^* - d_1}{v_c}) & si\ x = move \\ (0 \bullet ((q \bullet d_1) + e \cdot v), v, \frac{l^* - d_1}{v}) & en\ otro\ caso \end{cases}$$, donde $l^* = \begin{cases} l\ si\ d_1 < l \\ l + \Delta l\ \ en\ otro\ caso \end{cases}$

$$l(q \bullet d_1, v, s) = \begin{cases} leave & si\ d_1 \geq l \\ detect & en\ otro\ caso \end{cases}$$
$ta(q, v, s) = s$

Ejemplo 10: (ejercicio). Modificar el modelo del ejemplo anterior suponiendo ahora que la velocidad de la cinta depende (con alguna función conocida) del número de piezas que hay sobre la misma.

Mas adelante, retomaremos este ejemplo e iremos agregando las restantes partes del sistema de producción mencionado. La descripción detallada de dicho sistema, puede encontrarse en Lewerentz et al.(1995).

4. Modelos de Acoplamiento DEVS.

Hasta aquí hemos visto *especificaciones de comportamiento* DEVS. En las definiciones previas, mencionamos que los modelos podían también describirse a partir de su *estructura interna*, o sea, especificando el comportamiento de ciertos componentes básicos y la forma en que estos interactúan entre sí.

Desde el punto de vista de la tarea de modelado, las *especificaciones estructurales*, constituyen una simplificación significativa, ya que es extremadamente difícil poder hacer una descripción del comportamiento completo de un sistema relativamente complejo. Mucho mas fácil es describir el comportamiento de varios componentes elementales y luego especificar como interactúan entre sí.

Para DEVS, vamos a ver básicamente dos tipos de acoplamiento: Acoplamiento *modular* y acoplamiento *no modular*. En el primero, la interacción entre componentes será únicamente a través de las entradas y salidas de los mismos, mientras que en el segundo, la interacción se producirá a través de los estados. La conveniencia de uno u otro enfoque dependerá, como veremos, de la aplicación en cuestión.

4.1. Acoplamiento modular DEVS.

Un modelo de sistemas DEVS acoplados se define a partir de la siguiente estructura:

$N = \langle X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle$, donde

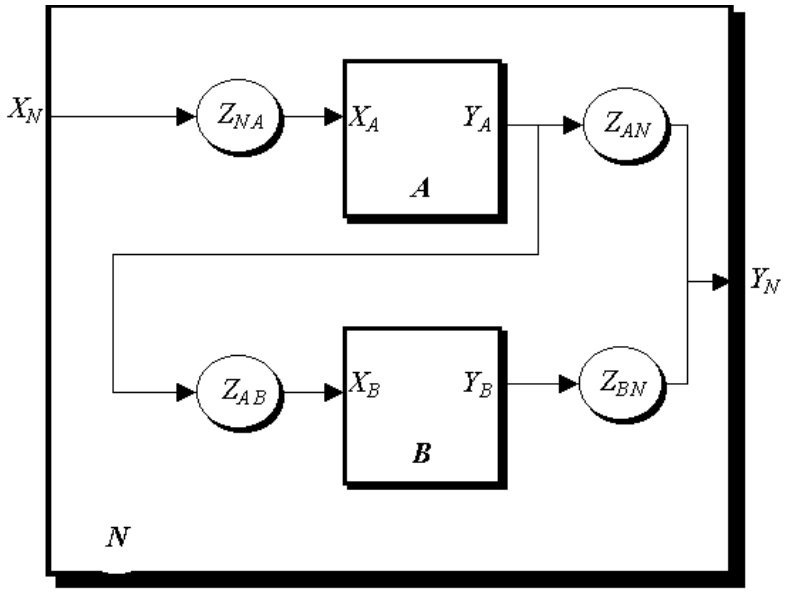
X es el conjunto de valores de entrada (del sistema acoplado).
Y es el conjunto de valores de salida (del sistema acoplado).
D es el conjunto de referencias a los componentes.
Para cada $d \in D$, M_d es una estructura que define un modelo DEVS.
$M_d = \langle X_d, S_d, Y_d, \delta_{\text{int}\,d}, \delta_{ext\,d}, l_d, ta_d \rangle$
Para cada $d \in D \bigcup \{N\}$, I_d es el conjunto de *influyentes* sobre d: $d \subseteq D \bigcup \{N\}, d \notin I_d$
Para cada $i \in I_d$, $Z_{i,d}$ es la función de traducción de salidas, donde

$Z_{i,d} : X_N \rightarrow X_d$, si $i = N$.
$Z_{i,d} : Y_i \rightarrow Y_N$, si $d = N$.
$Z_{i,d} : Y_i \rightarrow X_d$, si $d \neq N$ y $i \neq N$.

Select es una función, *Select:* $2^D \rightarrow D$, que verifica *Select* (E) $\in E$. Esta función cumple el papel de *función de desempate* para el caso de eventos simultáneos.

La figura 4 presenta una descripción gráfica de este tipo de especificación.

La interpretación funcional del acoplamiento es la siguiente: Cada modelo DEVS funciona independientemente desde el punto de vista interno, pero recibe eventos de entrada provenientes de las salidas de otros componentes (o de la entrada global del sistema). Estos componentes que pueden provocarle eventos de entrada a un determinado componente se denominan *influyentes*. Generalmente los eventos que puede recibir un componente no coinciden en el tipo con los que pueden generar sus influyentes. Para adaptar estas conexiones entonces se utilizan las *funciones de traducción*.



$$I_A = \{N\},\ I_B = \{A\},\ I_N = \{A, B\}$$

Figura 4: Especificación de Sistemas Acoplados.

El único posible punto de conflicto es cuando dos o más componentes tienen prevista una transición interna para el mismo instante de tiempo. En este caso, la elección de uno u otro para realizar primero la transición en general modificará sustancialmente el funcionamiento del sistema ya que las transiciones internas provocan eventos de salida que a su vez provocan transiciones externas en los demás componentes. Evidentemente, el orden en que se produzcan dichas transiciones puede alterar el funcionamiento global del sistema.

La solución de este conflicto es efectuada a través de la función *Select*, que establece prioridades para las transiciones internas de diferentes componentes. Cuando un determinado subconjunto de componentes tiene agendada su transición interna de manera simultánea, la función *Select* aplicada a dicho subconjunto devuelve un elemento del mismo que será quien transicione en primer lugar.

Ejemplo 11: Supongamos que conectamos el procesador del Ejemplo 1 con la cola del Ejemplo 2 de forma tal que los trabajos lleguen a la cola y ésta los envíe al procesador a medida que el mismo se desocupe. La Figura 5 muestra el esquema del sistema acoplado:

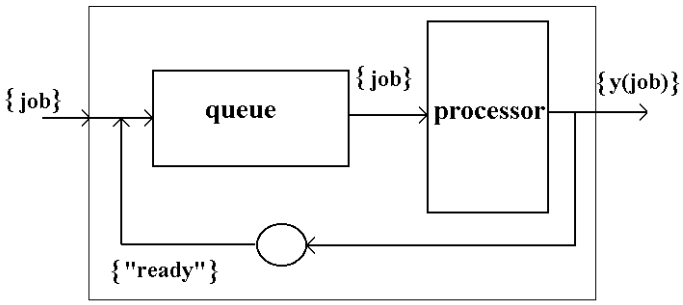


Figura 5. Acoplamiento de una cola y un procesador

El modelo DEVS de acoplamiento es el siguiente:

$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle$, donde:

$X = \{job_1, job_2, ..., job_n\}$

$Y = \{y(job_1), y(job_2), ..., y(job_n)\}$

$D = \{P, Q\}$

Las estructuras M_d son las definidas en los ejemplos 1 y 2.

$I_P = \{Q\}, I_Q = \{P, N\}, I_N = \{P\}$

$Z_{Q,P}(job) = job$, $Z_{P,Q}(y) =$"$ready$", $Z_{N,Q}(job) = job$, $Z_{P,N}(y) = y$

Ejemplo 12: Modificar el ejemplo anterior para especificar ahora un sistema en el cual los trabajos son efectuados por dos procesadores (con sus respectivas colas) en serie, como se indica en la Figura 6.

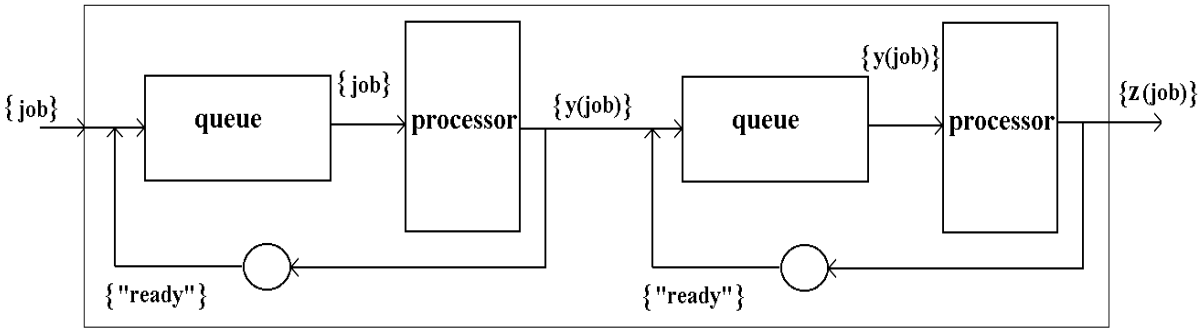


Figura 6. Acoplamiento de dos procesadores con buffers

4.2. Clausura bajo acoplamiento modular DEVS.

Una de las propiedades fundamentales del acoplamiento modular DEVS es la clausura. El cumplimiento de la propiedad de clausura nos garantiza que el acoplamiento de modelos DEVS nos define un nuevo modelo DEVS equivalente. Esto implica que un modelo DEVS acoplado puede utilizarse a su vez dentro de un modelo más complejo de acoplamiento, dando paso a lo que se denomina *jerarquía del acoplamiento*.

La posibilidad de acoplar modelos de manera *jerárquica* es lo que garantiza la reusabilidad de los mismos. Un modelo (posiblemente acoplado) realizado como parte de un modelo mas general, puede utilizarse en el contexto de otro modelo acoplado sin necesidad de realizar modificaciones.

Veremos entonces a continuación, de manera constructiva, como una expresión de acoplamiento DEVS define un modelo atómico DEVS, probando así la propiedad de clausura.

La estructura acoplada:

$N = \langle X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle$ de los subsistemas:

$M_d = \langle X_d, S_d, Y_d, d_{\text{int}\,d}, d_{ext\,d}, l_d, ta_d \rangle$

define un modelo DEVS atómico $M = \langle X, S, Y, d_{\text{int}}, d_{ext}, l, ta \rangle$ donde:

$X = X_N$, $Y = Y_N$,

$S = \underset{d \in D}{\times} Q_d$ con $Q_d = \{(s_d, e_d)\}$ donde $s_d \in S_d$ y e_d es el tiempo transcurrido desde la última transición del subsistema M_d hasta la última transición del sistema N $(e_d \geq 0)$. (Se dice que N tiene una transición cuando alguno de sus subsistemas tiene una transición).

La función de transición externa es:
$d_{ext}(s,e,x) = s'$, donde
$s = (...,(s_d, e_d),...)$, $s' = (...,(s'_d, e'_d),...)$ y además,
$\forall d / N \in I_d \wedge x_d \neq f$, $s^*_d = d_{ext\,d}(s_d, e_d + e, x_d)$ con $x_d = Z_{N,d}(x)$
y luego,

$$(s'_d, e'_d) = \begin{cases} (s^*_d, 0) & si \quad N \in I_d \wedge x_d \neq f \\ (s_d, e_d + e) & en\ otro\ caso \end{cases}$$

La función de avance de tiempo (o tiempo de vida) es:
$ta(s) = min\{s_d \mid d \in D\}$, donde $s_d = ta_d(s_d) - e_d$ (tiempo que falta para el próximo evento en M_d).

La función de transición interna es:
$d_{int}(s) = s'$, donde:
$s = (...,(s_d, e_d),...)$, $s' = (...,(s'_d, e'_d),...)$
Sea $IMM(s) = \{d \mid d \in D \wedge s_d = ta(s)\}$ el conjunto de inminentes (candidatos para la próxima transición), luego definimos:
$d^* = Select(IMM(s))$ (M_{d^*} es el subsistema que va a ejecutar su transición interna, ya que es el prioritario).

$$s'_d = \begin{cases} d_{int\,d^*}(s_{d^*},) & si \quad d = d^* \\ d_{ext\,d}(s_d, e_d + ta(s), x_d) & si \quad d^* \in I_d \wedge x_d \neq f \\ s_d & en\ otro\ caso \end{cases}$$

donde $x_d = Z_{d^*,d}(l_{d^*}(s_{d^*}))$, y

$$e'_d = \begin{cases} 0 & si \quad d = d^* \vee (d^* \in I_d \wedge x_d \neq f) \\ e_d + ta(s) & en\ otro\ caso \end{cases}$$

Por último, la función de salida es:

$$l(s) = \begin{cases} Z_{d^*,N}(l_{d^*}(s_{d^*})) & si \quad d^* \in I_N \\ f & en\ otro\ caso \end{cases}$$

Ejemplo 13: (ejercicio) Utilizar la idea del acoplamiento jerárquico para, partiendo del modelo obtenido en el Ejemplo 11, obtener el modelo del Ejemplo 12.

4.3. DEVS con puertos de entrada y salida.

La introducción de puertos de entrada y salida, puede simplificar bastante la tarea de modelado, especialmente en lo que se refiere al acoplamiento de modelos. En el formalismo DEVS, es posible introducir el concepto de puertos simplemente definiendo los conjuntos X e Y (conjuntos de valores de entrada y salida respectivamente) como sigue:

$X = \{(p,v) \mid p \in InPorts, v \in X_p\}$, siendo *InPorts* el conjunto de puertos de entrada y X_p el conjunto de valores de entrada en el puerto p.

$Y = \{(p,v) \mid p \in OutPorts, v \in Y_p\}$, siendo *OutPorts* el conjunto de puertos de salida y Y_p el conjunto de valores de salida en el puerto p.

Ejemplo 14: En la cola del Ejemplo 2, podemos definir dos puertos de entrada: por uno llegan los trabajos y por el otro llegan las señales “ready”. Así, el modelo queda:

$M_Q = \langle X, S, Y, d_{\text{int}}, d_{ext}, l, ta \rangle$, donde
$X = \{inport_1\} \times J \cup \{inport_2\} \times \{\text{"}ready\text{"}\}$, con $J = \{job_1, job_2, ..., job_n\}$
$S = J^+ \cup \{f\} \times \{0; \infty\}$
$Y = \{outport_1\} \times J$
$d_{int}(q \bullet job, s) = (q, \infty)$
$d_{ext}[q, s, e, (x, port)] = \begin{cases} (x \bullet q, \infty) & si\ port = inport_1 \\ (q, 0) & en\ otro\ caso \end{cases}$
$l(q \bullet job, s) = (outport_1, job)$
$ta(job, s) = s$

4.3.1. Acoplamiento DEVS mediante puertos.

La especificación formal del acoplamiento de modelos DEVS con puertos es la siguiente:

N = (X, Y, D, {M_d | d Î D}, EIC, EOC, IC, Select), donde

$X = \{(p,v) \mid p \in InPorts, v \in X_p\}$, siendo *InPorts* el conjunto de puertos de entrada y X_p el conjunto de valores de entrada en el puerto *p*.

$Y = \{(p,v) \mid p \in OutPorts, v \in Y_p\}$, siendo *OutPorts* el conjunto de puertos de salida y Y_p el conjunto de valores de salida en el puerto *p*.

D es el conjunto de referencias a los componentes;

Para cada *d* Î *D*, $M_d = (X_d, Y_d, S, d_{extd}, d_{intd}, l_d, ta_d)$ es un modelo *DEVS*, donde
$X_d = \{(p,v) \mid p \in InPorts_d, v \in X_{p_d}\}$
$Y_d = \{(p,v) \mid p \in OutPorts_d, v \in Y_{p_d}\}$

El acoplamiento externo de entrada *EIC* (*external input coupling*) conecta las entradas externas con las entradas de los componentes:
EIC Í {((N, ip_N), (d, ip_d)) | ip_N Î InPorts, d Î D, ip_d Î $InPorts_d$}

El acoplamiento externo de salida *EIC* (*external output coupling*) conecta las salidas de los componentes con las salidas externas:
EOC Í {((d, op_d), (N,op_N)) | op_N Î OPorts, d Î D, op_d Î $OPorts_d$}

El acoplamiento interno *IC* (*internal coupling*) conecta las salidas de los componentes con las entradas de los mismos:
IC Í {((a, op_a), (b, ip_b)) | a, b Î D, op_a Î $OutPorts_a$, ip_b Î $InPorts_b$}

Como antes, no se permite feedback directo, luego
((*d*, *opd*), (*e*, *ipd*)) ∈ *IC* implica *d* ¹ *e*.

Select: 2^D ® *D,* es igual que antes.

Ejemplo 15: Considerar un sistema formado por una cola como la del Ejemplo 14, y un procesador como el del Ejemplo 1, pero provisto de dos puertos de salida: en uno envía los trabajos y en el otro, la señal de "ready". El esquema es el de la Figura 7.

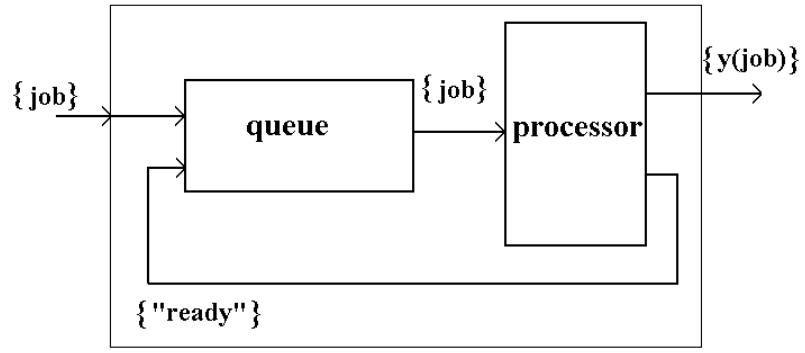


Figura 7. Un procesador y una cola acoplados mediante puertos

La especificación de acoplamiento queda:

$N = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select)$, donde
$X = J \times \{inport_1\}$
$Y = \{y(job) \mid job \in J\} \times \{outport_1\}$
$D = \{P, Q\}$
$EIC\{(N, inport_1), (Q, inport_1)\}$
$EOC\{(P, outport_1), (N, outport_1)\}$
$IC\{(P, outport_2), (Q, outport_2)\}$

Ejemplo 16 (ejercicio): Obtener el modelo del procesador con puertos del ejemplo anterior. (Observar que aparece una dificultad ligada a la necesidad de enviar simultáneamente las señales de "ready" y de salida del trabajo).

4.3.2. Jerarquía del acoplamiento DEVS mediante puertos.

Puede verificarse de forma bastante sencilla que el acoplamiento mediante puertos de entrada y salida es en realidad una particularización del acoplamiento modular descripto en la sección 4.1. De esta manera, la propiedad de clausura demostrada para el acoplamiento modular general es también válida para el acoplamiento mediante puertos. Mas aún, puede verse de manera simple que la definición realizada para el acoplamiento mediante puertos define un modelo DEVS atómico que además tiene puertos de entrada y salida. De esta forma, el acoplamiento mediante puertos puede también realizarse de manera jerárquica.

4.4. Acoplamiento DEVS no modular.

Definiremos ahora una especificación denominada DEVS multicomponentes. Se trata de una especificación *no modular* de interacción entre componentes. Como veremos, no se trata de acoplamiento de modelos DEVS, sino de acoplamiento de componentes, cuyo resultado puede ser visto como un modelo DEVS.

Un DEVS multicomponentes es una estructura:

multiDEVS = $\langle X, Y, D, \{M_d\}, Select \rangle$, donde

X, Y son los conjuntos de eventos de entrada y salida.
D es el conjunto de referencias a componentes
$Select: 2^D \rightarrow D$ con $Select(E) \in E$ es una función para arbitrar eventos simultáneos

Para cada $d \in D$, $M_d = \langle S_d, I_d, E_d, d_{ext,d}, d_{int,d}, \lambda_d, ta_d \rangle$, donde

S_d es el conjunto de estados secuenciales de d,

$Q_d = \{(s, e_d) \mid s \in S_d, e_d \in \Re\}$ es el conjunto de estados totales de d,

$I_d \subseteq D$ es el conjunto de componentes influyentes sobre d,

$E_d \subseteq D$ es el conjunto de componentes influidos por d,

$\delta_{ext,d}: \underset{i \in I_d}{\times} Q_i \times X \to \underset{j \in E_d}{\times} Q_j$ es la función de transición externa,

$\delta_{int,d}: \underset{i \in I_d}{\times} Q_e \to \underset{j \in E_d}{\times} Q_j$ es la función de transición interna,

$\lambda_d: \underset{i \in I_d}{\times} Q_i \to Y$ es la función de salida, y

$ta_d: \underset{i \in I_d}{\times} Q_i \to \Re^+_0 \cup \{\infty\}$ es la función de avance de tiempo.

El funcionamiento es el siguiente: Los eventos internos son agendados individualmente por cada componente $d \hat{I} D$ a través de las funciones de avance de tiempo individuales ta_d. Cuando un evento ocurre en uno de los componentes, este es ejecutado provocando cambios de estados determinados por la función de transición interna $d_{int,d}$ de dicho componente. Esta función depende del estado total de todos los componentes influyentes sobre d (el conjunto I_d), y aplica sobre todos los componentes influidos por d (el conjunto E_d), determinándoles su nuevo estado total $q_i = (s_i,e_i)$. La función *Select*, en tanto, es la que determina que componente ejecuta la función de transición interna cuando varios eventos en diferentes componentes son inminentes. Finalmente, los eventos externos son ejecutados por las funciones de transición externa individuales que estén definidas (tiene que haber coherencia entre dichas definiciones).

Ejemplo 17: "Juego de la Vida": Uno de los ejemplos mas difundidos de autómatas celulares es el denominado "Juego de la Vida" (Game of Life), presentado por Gardner en 1970. El mismo consiste en una disposición bidimensional de casilleros que conforman una cuadrícula. Considerado cada casillero como una célula, esta puede tener dos estados: viva o muerta. El vecindario de cada célula está conformado por las cuatro células adyacentes lateralmente mas las cuatro células adyacentes de manera diagonal. Las reglas de evolución son las siguientes:

- Si una célula viva tiene dos o tres vecinas vivas entonces sobrevive. Caso contrario muere, ya sea por aislamiento o por superpoblación.

- Si el vecindario de una célula muerta consta de tres células vivas entonces se produce un nacimiento. Caso contrario, la célula permanece muerta.

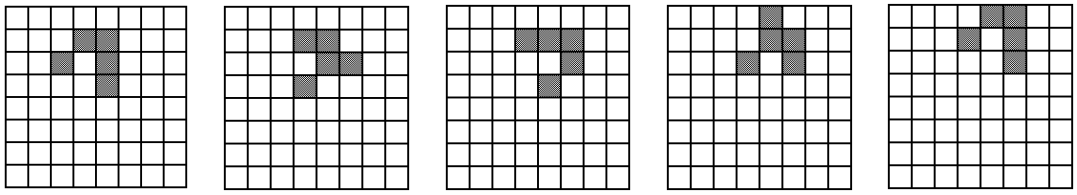


Figura 8: Una posible trayectoria del Juego de la Vida

Consecuentemente, cada célula puede pensarse como un modelo de tiempo discreto, el cual consta de ocho entradas (las salidas de las células pertenecientes a su vecindario), una variable de estado que indica si está viva o muerta (podríamos suponer que es una variable numérica que vale 0 si está muerta y 1 si está viva), y una salida, que coincide con el estado.

La expresión de la transición del estado de una célula genérica sería la siguiente:

$$v_{ij}(k+1) = \begin{cases} 1 & si\ sum = 3 \vee (sum = 2 \wedge v_{ij}(k) = 1) \\ 0 & en\ otro\ caso \end{cases}$$

donde *sum* es la suma de los estados en el vecindario de v_{ij}.

Un programa que permita simular un modelo de este tipo, bajo un enfoque de tiempo discreto deberá simular cada célula a cada instante de tiempo para poder determinar su estado futuro. Esto es, deberá evaluar la función de transición de cada célula en cada iteración. Si suponemos que el sistema tiene dimensiones relativamente grandes, sin duda este tipo de simulación tendrá enormes desventajas en lo relativo al tiempo de cálculo.

Existe sin embargo una simplificación al problema de la simulación de este ejemplo. Si en el vecindario de una célula no se produjo ningún cambio en la n-ésima iteración, sin dudas en dicha célula no se producirá ningún cambio en la siguiente iteración (en tiempo n+1) y por consiguiente, cuando no se produzcan cambios en el vecindario no será necesario evaluar que ocurrirá en la célula en la siguiente iteración.

Esta forma de pensar puede capturarse mediante una especificación DEVS multicomponente (multiDEVS) en la cual cada componente represente una célula, y cada vez que una célula transicione cambiando su valor, provoque cambios en el tiempo de vida de las células vecinas de forma tal de forzar que dichas células evalúen su estado en el siguiente paso.

El modelo MultiDEVS puede ser el siguiente:

$N = \langle X, Y, D, \{M_d\}, Select \rangle$, con

$X = Y = \{f\}$ (no hay entradas ni salidas, es un sistema autónomo)

$D = \{(i, j) \mid 1 \le i \le n, 1 \le j \le m\}$

$M_d = \langle S_d, I_d, E_d, d_{\text{int}\,d}, d_{ext\,d}, l_d, ta_d \rangle$, donde

$S_d = \{0,1\} \times \{0,1\} \times Z_0^+ \infty$ (en el estado incluimos el estado actual de la célula, el anterior y el tiempo de vida, que será un número entero si consideramos que el intervalo entre dos estados consecutivos es entero.

$I_d = I_{i,j} = \{(k,l) \in D / |k - i| \le 1 \wedge |l - j| \le 1\}$

$E_d = I_d$

$d_{\text{int}\,d}(...,(s_i, e_i),...) = (...,(s'_i, e'_i),...)$, donde $s_i = (v_i, v_{ant\,i}, s_i)$ y $s'_i = (v'_i, v'_{ant\,i}, s'_i)$, siendo

$$e'_i = \begin{cases} e_i & si\ i \ne d \\ 0 & en\ otro\ caso \end{cases}$$

$$v'_i = \begin{cases} v_i & si\ i \ne d \\ 1 & si\ i = d \wedge [(v_d = 1 \wedge 2 \le \Sigma \le 3) \vee (v_d = 0 \wedge \Sigma = 3)], \\ 0 & en\ otro\ caso \end{cases}$$

$$\text{donde } \Sigma = \sum_{i \in I_d / i \ne d} \bar{v}_i \text{, con } \bar{v}_i = \begin{cases} v_i & si\ e_i > 0 \\ v_{ant\,i} & en\ otro\ caso \end{cases}$$

$$v'_{ant\,i} = \begin{cases} v_{ant\,i} & si\ i \ne d \\ v_i & en\ otro\ caso \end{cases}$$

$$s'_i = \begin{cases} s_i \quad si\ i \neq d \wedge v_d = v'_d \\ \min(s_i, \Delta t) \quad si\ i \neq d \wedge v_d \neq v'_d \\ \Delta t \quad si\ i = d \wedge \exists j \neq d\ /(v_j \neq v_{ant\ j} \wedge e_j = 0) \\ \infty \quad en\ otro\ caso \end{cases}$$

$ta_d(..., (s_i, e_i), ...) = s_d$

Debido a que no hay entradas ni salidas, las funciones de salida y de transición externa no están definidas.

4.5. Modelo atómico DEVS definido por un MultiDEVS.

Dado que en MultiDEVS no estamos acoplando modelos DEVS atómicos, sino componentes no modulares, no podremos hablar de clausura. Sin embargo, veremos que un MultiDEVS se comporta externamente como un modelo DEVS atómico. Esto nos permitirá utilizar MultiDEVS dentro de estructuras de acoplamiento modulares (como las de la sección 4.1).

Dado un MultiDEVS definido como en la sección 4.4, asociamos el siguiente modelo atómico:

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

X, Y coinciden con los del MultiDEVS mencionado.

$S = \times_{d \in D} Q_d$.

$ta(s) = \min \{\sigma_d \mid d \in D\}$ donde $\sigma_d = ta_d(..., q_i, ...) - e_d$ es el tiempo hasta el próximo evento agendado en d.

Definimos entonces el próximo componente agendado como:
$d^* = Select\ (\{d \mid \sigma_d = ta(s)\})$.

En base a esto, la función de transición interna queda:

$\delta_{int}((s_1, e_1), (s_2, e_2), ..., (s_n, e_n)) = ((s'_1, e'_1), (s'_2, e'_2), ..., (s'_n, e'_n))$ con
$(s'_j, e'_j) = (s_j, e_j + ta(s))$ if $j \notin E_{d^*}$
$(s'_j, e'_j) = \delta_{int,d^*}((..., (s_i, e_i + ta(s)), ...)).j$ if $j \in E_{d^*}$

con $i \in I_{d^*}$,.

La salida del sistema queda definida por la salida de d^*
$\lambda(s) = \lambda_{d^*}(..., q_i, ...)$, $i \in I_{d^*}$.

La función de transición externa δ_{ext} bajo ocurrencia de un evento de entrada x se define como el producto cartesiano de las funciones de transición externas $\delta_{ext,d}((..., q_i, ...), e, x)$ de todos los componentes $d \in D$. Definimos la función de transición externa global como:

$\delta_{ext,d}(((s_1, e_1), (s_2, e_2), ..., (s_n, e_n)), e, x) = ((s'_1, e'_1), (s'_2, e'_2), ..., (s'_n, e'_n))$, con
$(s'_j, e'_j) = (s_j, e_j + e)$ si $j \notin E_d$
$(s'_j, e'_j) = \delta_{ext,d}((..., (s_i, e_i + e), ...)).j$ si $j \in E_d$.

4.6. Modularización.

En los sistemas multicomponentes con acoplamiento no modular, cada componente puede leer y escribir directamente sobre el estado de otros componentes. En ciertas aplicaciones, debido a las características de reusabilidad y a las posibilidades de paralelización de los sistemas con acoplamiento modular, podría ser deseable transformar la descripción originalmente no modular en una modular. Como veremos, a través del siguiente procedimiento, esto es siempre posible.

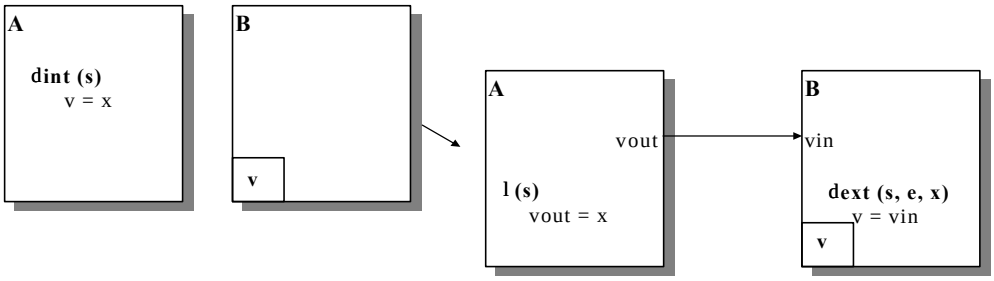


Figura 9: Modularización frente al acceso de escritura.

En primer lugar, debemos distinguir los dos tipos de acoplamiento no modular que se pueden presentar. La Figura 9 muestra el primer caso, en el cual la componente *A* tiene acceso de escritura a la variable de estado *v* de la componente *B*. Esta situación puede ser modularizada agregando en *A* un puerto de salida, *vout*, y a *B*, un puerto de salida *vin*. Luego, se deben conectar ambos puertos y cada vez que *A* quiera escribir sobre *v*, deberá generar un evento de salida a través del puerto *vout* con el valor deseado. A través del acoplamiento, este evento será visto por *B* como un evento de entrada en el puerto *vin*. La función de transición externa de *B* deberá entonces modificar la variable *v* acorde al evento recibido en *vin*.

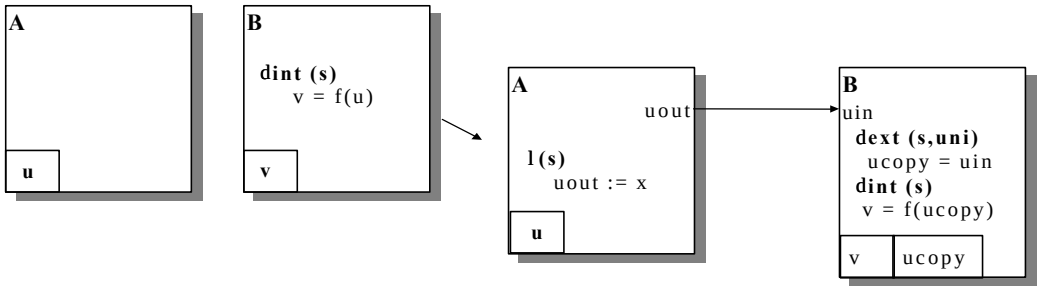


Figura 10: Modularización frente al acceso de lectura.

En el otro caso (Figura 10), la componente *B* tiene acceso de lectura a una variable *u* de la componente *A*. Esta situación es algo mas complicada, dado que *B* debe conocer siempre el valor actual de *u*. Para esto, la componente *B* deberá guardar una copia de *u* en su propio estado y cada vez que en *A* haya un cambio en el valor de dicha variable, *B* deberá ser informada del mismo. Para esto, deberá haber un puerto de salida en *A* y otro de entrada en *B*, de manera similar al caso anterior.

5. Ejemplo de Modelado de un Sistema de Producción.

Nos concentraremos ahora en el Sistema de Producción detallado en Lewerentz et al.(1995). El mismo consiste en una cinta transportadora que lleva piezas hasta una mesa móvil. Esta a su vez eleva las piezas para que puedan ser tomadas por un robot que las conduce hasta una prensa. Una vez que las piezas son prensadas, el robot las retira y las coloca en otra cinta transportadora que las conduce hasta el lugar donde se encuentra una grúa.

En el modelo planteado en la referencia, esta grúa coloca nuevamente las piezas en la primer cinta transportadora de forma de obtener un proceso cíclico (esto está hecho así ya que se trata de la descripción de un prototipo experimental). Nosotros consideraremos que una vez que las piezas son tomadas por la grúa salen de nuestro sistema. Asimismo, el arribo de piezas a la primer cinta será dictado por eventos de entrada al sistema global.

Naturalmente, además de los dispositivos indicados, existe un sistema de control que realiza el comando de todas estas operaciones. También hay toda una serie de sensores que le indican al sistema de control la ocurrencia de eventos en los dispositivos.

Para obtener un modelo DEVS del sistema completo, iremos detallando modelos de los diferentes dispositivos y por último haremos una especificación de acoplamiento modular de los mismos. Si bien está detallado el funcionamiento de cada dispositivo en la referencia mencionada, no tenemos allí una descripción del sistema de control, por lo que además de modelar cada dispositivo, deberemos diseñar dicho sistema.

5.1. Modelo de las cintas transportadoras:

(Ver ejemplos 8 y 9)

5.2. Modelo de la mesa móvil:

La mesa móvil cuenta con dos posibles tipos de movimientos: un movimiento de rotación y otro de elevación. La mesa cuenta con topes que le impiden moverse mas allá de las posiciones correctas para recibir o enviar piezas, de forma tal que cuando se alcanza la posición correcta, la mesa se detiene sola. Consideraremos también que ambos movimientos no pueden realizarse simultáneamente. Para poder recibir piezas provenientes de la cinta transportadora, la mesa deberá estar en la posición inferior y rotada hacia la izquierda, mientras que para que el robot pueda tomar piezas de la mesa, esta deberá estar en su posición superior y rotada hacia la derecha. Con esto, los eventos que puede recibir la mesa son los siguientes:

- "*rotateright*" y *"rotateleft*" (ambos sentidos de rotación).
- "*moveup*" y "*movedown*" (arriba y abajo).
- "*arrive*" cuando llega una pieza.
- "*pick*" cuando el robot intenta tomar una pieza.

A su vez, saldrán los siguientes eventos:

- "*up*", "*down*" cuando alcance las posiciones superior e inferior respectivamente.
- "*right*","*left*", cuando alcance las rotaciones hacia la derecha y hacia la izquierda.
- "*picked*" cuando el robot toma una pieza.

Con todas estas consideraciones, un posible modelo es el siguiente:

$M_M = \langle X, S, Y, d_{int}, d_{ext}, l, ta \rangle$, donde

$X = \{arrive, pick, moveup, movedown, rotateright, rotateleft\}$

$S = \Re^4 \times \{charged, empty\} \times \{picking, notpicking\} \times \Re_0^+$, donde los cuatro números reales representan la posición y velocidad vertical y angular.

$Y = \{up, down, right, left, picked\}$

Denominamos a_r y a_l a las posiciones extremas derecha e izquierda de la mesa, mientras que h_u y h_d son las posiciones extremas superior e inferior respectivamente.

$$d_{int}(h,v,a,w,s_1,s_2,s) = \begin{cases} (h,v,a,w,empty,notpicking,s') & si\ s_2 = picking \\ (h + s\cdot v, 0, a + s\cdot w, 0, s_1, s_2, \infty) & en\ otro\ caso \end{cases}, \text{ donde } s' = \begin{cases} \frac{h_u - h}{v_u} & si\ v = v_u \\ \frac{h_d - h}{v_d} & si\ v = v_d \\ \frac{a_r - a}{w_r} & si\ w = w_r \\ \frac{a_l - a}{w_l} & si\ w = w_l \\ \infty & en\ otro\ caso \end{cases}$$

$$d_{ext}(h,v,a,w,s_1,s_2,s,e,x) = \begin{cases} (h', v_u, a', 0, s_1, s_2, \frac{h_u - h'}{v_u}) & si\ x = moveup \\ (h', v_d, a', 0, s_1, s_2, \frac{h_d - h'}{v_d}) & si\ x = movedown \\ (h', 0, a', w_r, s_1, s_2, \frac{a_r - a'}{w_r}) & si\ x = rotateright \\ (h', 0, a', w_l, s_1, s_2, \frac{a_l - a'}{w_l}) & si\ x = rotateleft \\ (h', v, a', w, charged, s_2, s') & si\ x = arrive \wedge s_1 = empty \wedge h' = h_d \wedge a' = a_l \\ (h', v, a', w, s_1, picking, 0) & si\ x = pick \wedge s_1 = charged \wedge h' = h_u \wedge a' = a_r \\ (h', v, a', w, s_1, s_2, s') & en\ otro\ caso \end{cases}$$

donde $h' = h + v\cdot e$, $a' = a + w\cdot e$, y $s' = \begin{cases} \frac{h_u - h'}{v_u} & si\ v = v_u \\ \frac{h_d - h'}{v_d} & si\ v = v_d \\ \frac{a_r - a'}{w_r} & si\ w = w_r \\ \frac{a_l - a'}{w_l} & si\ w = w_l \\ \infty & en\ otro\ caso \end{cases}$

$$l(h,v,a,w,s_1,s_2,s) = \begin{cases} picked & si\ s_2 = picking \\ up & si\ v = v_u \\ down & si\ v = v_d \\ right & si\ w = w_r \\ left & si\ w = w_l \end{cases}$$

$$ta(h,v,a,w,s_1,s_2,s) = s$$

Ejercicio: Especificar cuales son las entradas y las salidas del Robot, de la Prensa y del Sistema de Control, y obtener el modelo de acoplamiento del sistema completo.

6. Simulación de Modelos DEVS.

Plantearemos simuladores DEVS utilizando un esquema que nos permita explotar la construcción jerárquica y modular de dichos sistemas. Para eso, definiremos diferentes clases de simuladores asociados a cada tipo de especificación:

- Para la especificación básica (atómica), es decir, DEVS = $\langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$, tendremos asociada una clase que denominaremos *simulador*.
- Para la estructura de modelos acoplados (modular), tendremos asociada una clase de simulador que denominaremos *coordinador*.

Para seguir con el lineamiento de construcción jerárquica, naturalmente cada simulador deberá utilizar un protocolo genérico de mensajes que le permita coordinar con los otros para ejecutar la simulación.

La figura 11 muestra como un modelo jerárquico tiene asociado un simulador abstracto. El coordinador raíz (*root-coordinator*) es el encargado de iniciar los ciclos de simulación y de dirigir el avance del tiempo de la simulación.

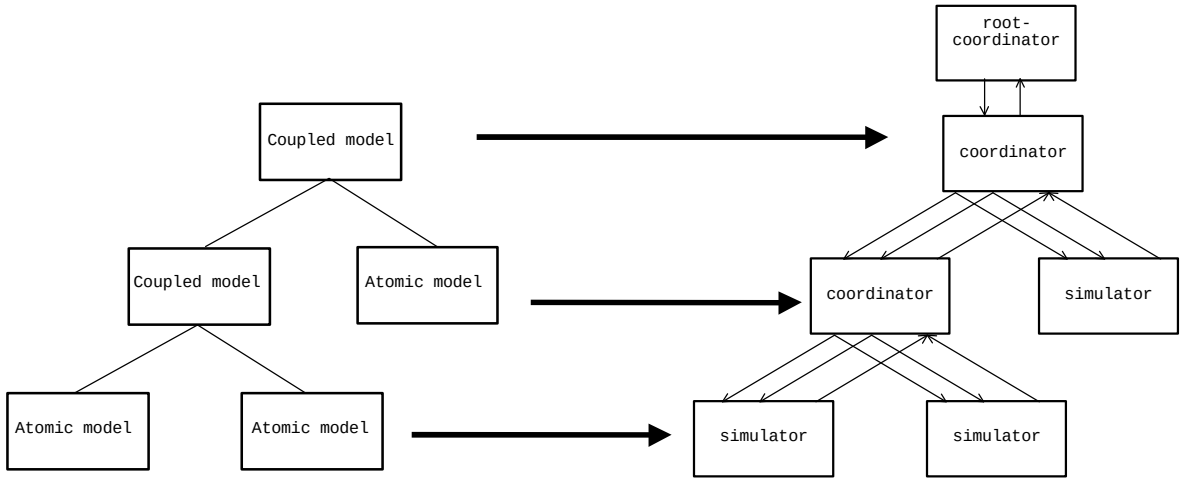


Figura 11: Simulador jerárquico asociado a un modelo jerárquico.

La idea básica de la simulación de modelos DEVS, al igual que la de cualquier paradigma de eventos discretos, es la formulación de una lista de eventos y la ejecución de los mismos de manera secuencial, provocando transiciónes de estado. Naturalmente, cada vez que se ejecute algún evento, se agendarán nuevos eventos en la lista.

Un simulador jerárquico para DEVS jerárquicos consiste de *simuladores* (*devs_simulators*) y *coordinadores* (*devs_coordinators*). Un mensaje de inicialización (*i,t*) es enviado en el instante de inicialización por cada *padre* a todos sus subordinados. El próximo evento interno agendado es llevado a cabo a través del mensaje de transición de estados (**,t*) enviado por el *coordinador* a su *hijo* inminente. En este caso, un mensaje de salida (*y*,*t*) es enviado por el *hijo* al *coordinador*. Los mensajes de entrada, en tanto, son enviados por el *coordinador* a sus subordinados para causarles eventos externos. La Figura 12 muestra los mensajes que se intercambian entre un coordinador y sus hijos.

La idea entonces es que cada coordinador decide quien de sus hijos va a transicionar. Esto requiere entonces que el coordinador conozca cual es el tiempo de próximo evento de cada uno de sus hijos. De esta forma, cada modelo debe tener una propiedad *tn* que es el tiempo del próximo evento interno, que debe ser conocida por su padre. Al ser cada coordinador a su vez

hijo de otro coordinador (eventualmente el coordinador raiz), deberá tener también esta propiedad *tn* que será lógicamente igual al menor de los *tn* de sus hijos.

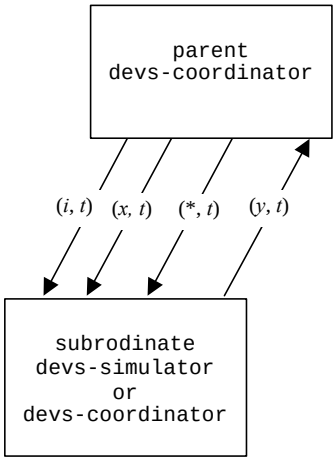


Figura 12: Protocolo de simulación DEVS

6.1. Simulador asociado a un DEVS atómico (*DEVS- Simulator*)

El *devs-simulator* utiliza básicamente dos variable de tiempo: *tl* (tiempo del último evento ocurrido) y *tn* (tiempo del próximo evento agendado). De la definición de la función de avance de tiempo, sigue:

$tn = ta(s) + tl$

En tanto, si conocemos el tiempo global de simulación (t), es posible determinar el tiempo transcurrido en el estado:

$e = t - tl$

y el tiempo para el próximo evento:

$s = tn - t = ta(s) - e.$

El tiempo para el próximo evento, *tn*, como ya mencionamos es conocido por el *padre* para permitir la correcta sincronización. En base a todo lo dicho, el algoritmo coorespondiente a la clase *devs-simulator* es el que sigue:

```
Devs-simulator
  variables:
    parent     // parent coordinator
    tl       // time of last event
    tn       // time of next event
    DEVS       // associated model
        with total state (s, e)
    y        // current output value of the associated model
  when receive i-message  (i, t)  at time t
    tl = t - e
    tn = tl + ta(s)
  when receive *-message (*, t) at time t
    if  t != tn then
      error: bad synchronization
```

```
      y = λ(s)
      send y-message (y, t) to parent coordinator
      s = δ_int (s)
      tl = t
      tn = tl + ta(s)
    when receive x-message (x, t)at time t with input value x
      if  not (tl ≤ t ≤ tn)  then
        error: bad synchronization
      e = t - tl
      s = δ_ext (s, e, x)
      tl = t
      tn = tl + ta(s)
end Devs-Simulator
```

Algoritmo 1: Simulador para DEVS atómicos

Es simple demostrar que el Algoritmo 1 simula correctamente el sistema DEVS especificado.

6.2. Simulador asociado a modelos DEVS acoplados (*DEVS-coordinator*)

Tenemos ahora una estructura del tipo $N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}\ Select \rangle$. Cada componente M_d es simulada por su propio simulador (*devs-simulator* o *devs-coordinator* según sea o no un modelo atómico). El *coordinador* asociado a *N* deberá utilizar el protocolo de sus subordinados y brindar el mismo protocolo a su *padre*. De esta forma podrá coordinar correctamente los componentes y manipular las entradas que lleguen a *N*.

El *coordinador*, básicamente, mantiene una lista de eventos consistente en pares compuestos por el tiempo de próximo evento (tn_d) de cada subordinado y la referencia al mismo. La lista está ordenada en función de dichos tiempos y en caso de igualdad, de acuerdo a la función *Select*. De esta forma, su propio tiempo de próximo evento será:

$tn = min\ \{tn_d \mid d \in D\}$

Este tiempo, es provisto por el *coordinador* a su *padre*. De manera análoga, el tiempo de último evento del coordinador, puede ser calculado (esto puede ser necesario en la inicialización, si se quiere implementar una detección de errores de sincronización) como:

$tl = \max\ \{tl_d \mid d \in D\}$

Además de todo esto, el *coordinador* deberá manejar la llegada de eventos de entrada al sistema y los eventuales eventos de salida de las componentes. De esta manera, recibirá de su padre, además del mensaje de inicialización (*i*, *t*), mensajes (*, *t*) y (*x*, *t*), indicando respectivamente transiciones internas y externas. En el primer caso, retransmitirá a sus hijos el mensajes de inicialización recibido.

Cuando reciba un mensaje de transición interna, se lo enviará a su *hijo* inminente (el primer componente de la lista ordenada), mientras que cuando reciba un mensaje de entrada (*x,t*) utilizará el mapa de interfaces para enviar a los subordinados que sean influenciados por la red (*N*) el mensaje correspondiente (x_d , *t*) donde x_d será calculado de acuerdo a la función $Z_{N,d}$.

En tanto, cuando reciba desde un hijo un mensaje de salida (y_{d*}, *t*), utilizará nuevamente el mapa de interfaces para enviar a sus subordinados influenciados por *d** mensajes de entrada

(calculados por $Z_{d^*,d}$) y si la red es influenciada por d^* enviará a su *padre* un mensaje de salida (y, t) calculado según $y = Z_{d^*,N}$.

Con todo esto, se plantea el siguiente algoritmo:

```
Devs-coordinator
variables:
  DEVN = (X, Y, D, {M_d}, {I_d}, {Z_i,d}, Select) //the associated
  network
  parent     // parent coordinator
  tl       // time of last event
  tn       // time of next event
  event-list   // list of elements (d, tn_d) sorted by tn_d
                 and Select
  d*       // selected imminent child
when receive i-message  (i, t)   at  time t
  for-each d in D do
    send i-message (i, t) to child d
  sort event-list according to tn_d and Select
  tl = max {tl_d | d ∈ D}
  tn = min {tn_d | d ∈ D}
when receive *-message  (*, t)  at  time t
  if  t != tn then
    error: bad synchronization
  d* = first (event-list)
  send *-message (*, t)  to d*
  sort event-list according to tn_d and Select
  tl = t
  tn = min {tn_d | d ∈ D}
when receive x-message (x, t)at time t with external input x
  if  not (tl ≤ t ≤ tn)  then
    error: bad synchronization  //consult external input
         coupling to get children influenced by the input
  receivers = {r | r ∈ D, N ∈ I_r , Z_N,r(x) ≠ Φ}
  for-each r in receivers
    send x-messages (x_r, t) with input value x_r= Z_N,r(x) to r
  sort event-list according to tn_d and Select
  tl = t
  tn = min { tn_d | d ∈ D}
when receive y-message (y_d*, t) with output y_d*  from d*
  if  d* ∈ I_N & Z_d*,N(y_d*) ≠ Φ then
    send y-message (y_N, t)  with value y_N = Z_d*,N(y_d*) to parent
  receivers = {r | r ∈ D, d* ∈ I_r , Z_d*,r(y_d*) ≠ Φ}
  for-each r in receivers
    send x-messages (x_r, t) with input value x_r = Z_d*,r(y_d*) to
    r
end Devs-coordinator
```

Algoritmo 2: Simulador para DEVS acoplados

Vimos en el capítulo anterior que una especificación de modelos DEVS acoplados definía un nuevo modelo atómico DEVS (clausura bajo composición).
Es posible demostrar que el Algoritmo 2 implementa de manera correcta la especificación DEVS atómica definida por el acople y que coordina correctamente a sus subordinados. De

esta manera, queda garantizado que un *devs-coordinator* puede ser tratado como un simulador para modelo atómico, garantizando esto la posibilidad de construcción jerárquica.

6.3. Coordinador Raíz (*Root-Coordinator*)

Resta finalmente definir el coordinador raíz, que es quien implementa el ciclo global de simulación. El mismo cuenta con un único subordinado consistente en el simulador del sistema global.

El funcionamiento es el siguiente: envía un mensaje de inicialización a su *hijo* y luego entra en un ciclo en el cual redefine constantemente el tiempo global de simulación (*t*) enviando mensajes de transición al mismo.

```
devs-root-coordinator
  variables:
    t      // current simulation time
      child // direct subordinate devs-simulator or devs–
                coordinator

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

end devs-root-coordinator
```

Algoritmo 3: DEVS-Root-Coordinator

Como puede observarse de la especificación del coordinador raiz, el mismo no envía eventos externos al modelo global. Esto implica que en este esquema de simulación el modelo global no puede recibir eventos de entrada, por lo que el modelo global deberá incluir subsistemas encargados de generar tales entradas. De la misma forma, habrá también subsistemas "sumideros" que reciban los eventos considerados salidas del sistema en cuestión.

6.4. Simulador para DEVS Multicomponente (no modular)

Vimos en el capítulo 4 cómo una especificación multiDEVS definía un sistema atómico DEVS. Es interesante entonces, construír un simulador para multiDEVS que pueda ser tratado externamente como un simulador para DEVS atómico, y por lo tanto, pueda ser acoplado con otros simuladores como parte del esquema de simulación jerárquica visto anteriormente.

El simulador que veremos presenta un funcionamiento similar al del coordinador visto para sistemas acoplados, en el sentido que mantiene una lista de eventos agendados de la misma forma que el *coordinador* y que utiliza las mismas variables *tn* y *tl* propias. La diferencia radica en que es el mismo simulador el que ejecuta las transiciones de estado. De esta manera, no tiene hijos. El mismo simulador está asociado al modelo y a sus componentes.

Cuando el simulador (que denominaremos *Esdevs-simulator*) recibe el mensaje de inicialización (*i*, *t*), calcula los tiempos tl_d y tn_d de los componentes según:

$tl_d = t - e_d$

$tn_d = tl_d + ta_d((..., q_i, ...)))$,

donde (..., q_i, ...) es el vector de estados formado por los estados de los componentes influyentes sobre *d*.

Luego, los pares (d, tn_d) son insertados y ordenados en la lista de eventos al igual que antes (acorde a los tiempos y eventualmente a la función *Select*). De la misma forma, *tn* es calculado como el mínimo de los tn_d .

Cuando se recibe un mensaje de transición (*, *t*), se ejecuta la función de transición interna del primer componente de la lista (*d**), resultando en una modificación de los estados de todos los componentes *d* influidos por *d**, y por ende, en la modificación de los tiempos tn_d y tl_d correspondientes. Esto implicará un reordenamiento de la lista de eventos y un nuevo cálculo de los tiempos *tn* y *tl*.

Al recibir mensajes de entrada (*x*, *t*), son ejecutadas las transiciones externas de todos los componentes afectados por la entrada. Al igual que antes, esto traerá aparejado un nuevo reordenamiento de la lista y un nuevo cálculo de los tiempos *tn* y *tl*.

El algoritmo resultante, entonces, es el siguiente

```
Esdevs-simulator
variables:
  parent     // parent coordinator
  tl       // time of last event
  tn       // time of next event
  ESDEVS = (X, Y, D, {Md}, Select)
        with components Md = (Sd, Id, Ed, δext,d, δint,d, λd, tad)
  with states qd = (sd, ed)
           time of last event tld and time of next event tnd
           and local outputs yd
        event-list = list of pairs (d, tnd) sorted by tnd and
Select
when receive i-message  (i, t)  at time t
  for all components d ∈ D
    tld = t – ed
    tnd = tld + tad((..., qi, ...))
  for all components d ∈ D
    sort (d, tnd) into event-list
  tl = max {tld | d ∈ D}
  tn = min {tnd | d ∈ D}
when receive *-message (*, t) at time t
  if  t != tn then
    error: bad synchronization
    (d*, tnd*) = first (event-list)
  for all components i ∈ Id*
    ei = t - tli
  (..., qj, ...) = δint,d*((..., qi, ...))
  for all components j ∈ Ed*
    tlj = t - ej
    tnj = tlj + taj((..., qi, ...))
      remove (j, tnj) and
      reinsert (j, tnj) into event-list
  tl = t
  tn = min {tnd | d ∈ D}
```

```
when receive x-message (x, t) at time t  with input value x
  if  not (tl ≤ t ≤ tn)  then
    error: bad synchronization
  for all components d ∈ D if defined δ_ext,d
    for all components i ∈ I_d
      e_i = t - tl_i
    (..., q_j, ...) = δ_ext,d((..., q_i, ...), t-tl_d, x)
    for all components j ∈ E_d
      tl_j = t - e_j
      tn_j = tl_j + ta_j((..., q_i, ...))
      remove (j, tn_j) and
      reinsert (j, t + tn_j) into event-list
  tl = t
  tn = min {tn_d | d ∈ D}
end Esdevs-Simulator
```

Algoritmo 4: Simulador para Event Scheduling MultiDEVS

Otra vez, es posible demostrar que este algoritmo realiza correctamente la simulación del sistema DEVS atómico asociado al modelo DEVS multicomponente definido. Esto implicará que este Simulador puede utilizarse como si fuera el simulador de un DEVS atómico y por lo tante puede ser utilizado como parte de una construcción jerárquica.

Ejemplo 18: (ejercicio) En el lenguaje de su preferencia, programe un simulador para el sistema del ejemplo 15. Tenga en cuenta que deberá agregar un modelo DEVS generador de los trabajos (puede utilizar alguna función aleatoria que admita el lenguaje de programación utilizado para generar los mismos). Como simplificación puede adoptar que cada trabajo es identificado por un número que es a su vez el tiempo que demanda su procesamiento, y que la salida es el mismo número.

Nota: Observar que el envio de mensajes entre simuladores y/o coordinadores puede reemplazarse por ejecución de métodos de los mismos.

Ejemplo 19: (ejercicio) Modificar el ejemplo anterior agregando un nuevo procesador (con su respectivo buffer) en paralelo, con ambos buffers recibiendo trabajos alternadamente mediante una llave. El modelo de la llave puede ser el que sigue:

$M_Q = \langle X, S, Y, d_{\text{int}}, d_{ext}, l, ta \rangle$, donde

$X = \{inport_1\} \times J$, con $J = \{job_1, job_2, ..., job_n\}$

$S = J \cup \{f\} \times \{0;1\} \times \{0; \infty\}$

$Y = \{outport_0, outport_1\} \times J$

$d_{int}(job, p, s) = (f, \overline{p}, \infty)$

$d_{ext}[(job, p, s), e, (x, port)] = (x, p, 0)$

$l(job, p, s) = (outport_p, job)$

$ta(job, p, s) = s$

Ejercicio: Comparar los modelos de los ejemplos 12 y 13. Decidir cual es mas eficiente para la simulación en base a los simuladores planteados. Concluir sobre las ventajas y desventajas de la simulación de modelos jerárquicos y/o modelos *chatos*.

7. Ejemplo de modelado de un sistema de ascensores

A continuación veremos un ejemplo sobre un sistema de ascensores, en el cual diseñaremos diferentes opciones de control.

Ejemplo 20: Modelo de un ascensor.

Consideremos un ascensor que se puede comandar por los siguientes eventos: *arriba*, *abajo*, *parar*, *abrir* (puerta) y *cerrar* (puerta). Cada piso a su vez cuenta con un sensor que indica la presencia del ascensor, de modo tal que la salida del sistema "ascensor" son los eventos producidos por dichos sensores, además de dos eventos que indican que cuando se cerró o abrió la puerta. El modelo puede ser el que sigue:

$M_A = \langle X, S, Y, d_{int}, d_{ext}, l, ta \rangle$, donde

$X = \{arriba, abajo, parar, abrir, cerrar\}$

$S = \Re \times \{-v_b, 0, v_s\} \times \{abriendo, cerrando, abierta, cerrada\} \times \Re_0^+$

$Y = \mathbf{N} \cup \{abierta, cerrada\}$

$$d_{int}(h, v, p, s) = \begin{cases} (h, v, abierta, \infty) & si\ p = abriendo \\ (h, v, cerrada, \infty) & si\ p = cerrando \\ (h + s \cdot v, v, \left|\frac{\Delta h}{v}\right|) & en\ otro\ caso \end{cases}$$

$$d_{ext}(h, v, p, s, e, x) = \begin{cases} (h + e \cdot v, 0, p, \infty) & si\ x = parar \\ (h + e \cdot v, v_s, p, \frac{(n+1)\Delta h - (h + e \cdot v)}{v_s}) & si\ x = subir \wedge p = cerrada \\ (h + e \cdot v, -v_b, p, \frac{(h + e \cdot v) - n\Delta h}{v_b}) & si\ x = bajar \wedge p = cerrada \\ (h, v, abriendo, 0)\ si\ x = abrir \wedge v = 0 \\ (h, v, cerrando, 0)\ si\ x = cerrar \wedge v = 0 \\ (h + e \cdot v, v, p, s - e) & en\ otro\ caso \end{cases}$$

donde $n = \text{int}\left[\frac{h + e \cdot v}{\Delta h}\right]$

$$l(h, v, p, s) = \begin{cases} abierta & si\ p = abriendo \\ cerrada & si\ p = cerrando \\ \text{int}\left[\frac{h + s \cdot v}{\Delta h}\right] & en\ otro\ caso \end{cases}$$

$ta(h, v, p, s) = s$

Ejemplo 21: (ejercicio): Modificar el ejemplo anterior para considerar que hay un número finito de pisos y que las acciones de abrir la puerta y cerrar la puerta no son instantáneas.

Ejemplo 22: (ejercicio): Obtener un modelo con puertos de entrada y salida del ejemplo anterior.

Ejemplo 23: (ejercicio) Diseñar un modelo DEVS de un controlador para el ascensor al cual lleguen eventos con números enteros, indicando que el ascensor debe cerrar la puerta, ir hasta dicho piso y luego abrir la puerta. Dicho controlador recibe como eventos de entrada además las salidas del ascensor del ejemplo 22. Además de las salidas para el ascensor, debe provocar una señal de salida "libre" cada vez que finalice una tarea y "ocupado" cada vez que comience otra. Suponer que los eventos de entrada indicando que se debe ir a un nuevo piso mientras se está yendo a otro son ignorados. Realizar el modelo con puertos.

Ejemplo 24: (ejercicio). Construir un modelo acoplado del controlador y el ascensor en el cual, las salidas globales del sistema sean las señales "listo" del controlador y las entradas, los requerimientos de ir a un determinado piso.

Ejemplo 25: (ejercicio). La figura 13 muestra un esquema acoplado con dos ascensores (podrían ser mas) con sus respectivos controles (como en el ejemplo 24). Especificar la expresión de dicho acoplamiento.

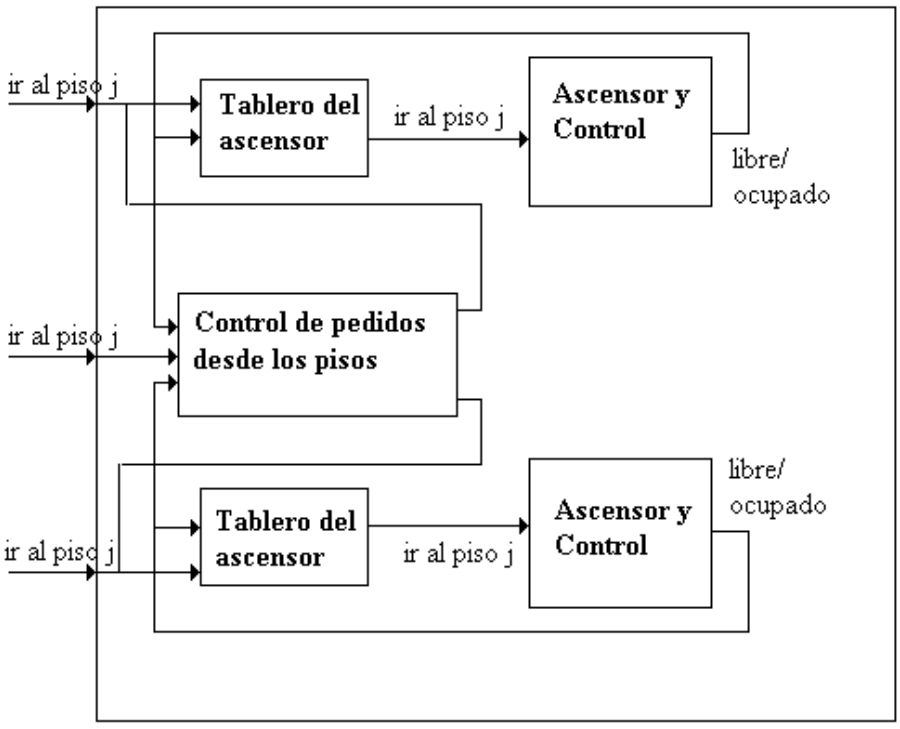


Figura 13: Esquema acoplado del control de ascensores

Ejemplo 26: (ejercicio). Obtener el modelo del tablero, suponiendo que consta de una cola en la que almacena pedidos (observar que no debe haber dos pedidos iguales, ya que no tiene sentido ir dos veces al mismo lugar). También debe tenerse en cuenta que podría ser conveniente no seguir el orden de llegada en los pedidos, ya que eso podría requerir subir y bajar demasiadas veces innecesariamente.

Ejemplo 27: (ejercicio). Diseñar un modelo para el control de pedidos desde los pisos. Suponer que al mismo llegan pedidos del tipo "ir al piso j" que indican que desde el piso j se pidió el ascensor. Dichos pedidos pueden ser enviados a uno u otro ascensor según alguna estrategia adoptada (puede ser que se alternen pedidos entre uno y el otro por ejemplo),

8. DEVS en simulación de sistemas continuos.

Una de las mas recientes aplicaciones de DEVS es la utilización de dicho formalismo para simular Sistemas Continuos. Básicamente encontramos dos enfoques: uno en el cual se busca aproximar las trayectorias continuas por secciones de polinomios (Giambiasi, 2001) y otro en el cual la aproximación se realiza a través de la cuantificación de las variables de estado del sistema (Kofman et al., 2001). El primer enfoque tiene inconvenientes en el trato de sistemas no lineales y además no constituye una transformación formal sobre el sistema continuno en la cual puedan estudiarse propiedades analíticas como convergencia (fundamental si uno intenta plantear un nuevo método numérico).

El segundo enfoque, en el planteo original de Zeigler (Zeigler et al, 1998,), tiene una falla debida a que la cuantificación realizada lleva a un modelo DEVS que puede ser no legitimado (o sea, no simulable). Este inconveniente es solucionado en (Kofman, 2000) y en (Kofman y Junco, 2001a) mediante la adición de histéresis en la cuantificación y la definición de *Quantized State Systems* (sistemas de estados cuantificados). De esta forma, además de garantizar simulabilidad, al ser una transformación formal, se demuestra también convergencia y estabilidad de las soluciones numéricas obtenidas.

Ambos enfoques cuentan también con trabajos de aplicación a Bond Graphs. Encuadrado en el primer enfoque, encontramos un método para la simulación de Bond Graphs lineales sin singularidades estructurales en (Naamane et al., 2001). Por su parte, la definición de Quantized Bond Graphs (Kofman y Junco, 2001b) corresponde con el segundo enfoque donde se permite la simulación de Bond Graphs generales y se demuestran las ventajas de la técnica frente a singularidades estructurales del tipo “causalidad derivativa”.

Nos concentraremos entonces en los métodos de simulación resultantes de la cuantificación con histéresis de los estados de un sistema continuo.

8.1. Sistemas de Estados Cuantificados.

Como mencionamos antes, la cuantificación en los sitemas de estados cuantificados (QSS por Quantized State Systems) es realizada mediante el uso de histéresis, Antes de definir QSS, el concepto de *Función de cuantificación con histéresis* será presentado. De aquí en mas, salvo que hagamos referencia a otro artículo, nos basaremos casi por completo en (Kofman y Junco, 2001a)

Sea $D=\{d_0,d_1,\cdots,d_n\}$ un conjunto de número reales donde $d_{i-1}<d_i$. Sea $x\in\Omega$ una trayectoria continua en la cual $x:\Re\to\Re$. Sea $b:\Omega\times\Re\to\Omega$ una función y asumamos que $q=b(x,t_0)$ satisface:

$$q(t)=\begin{cases} d_m & \text{if } t=t_0 \\ d_{i+1} & \text{if } x(t)=d_{i+1}\wedge q(t^-)=d_i\wedge i<r \\ d_{i-1} & \text{if } x(t)=d_i-e\wedge q(t^-)=d_i\wedge i>r \\ q(t^-) & otherwise \end{cases}, \text{ donde } m=\begin{cases} 0 & \text{if } x(t_0)<d_0 \\ r & \text{if } x(t_0)\geq d_r \\ j & \text{if } d_j\leq x(t_0)<d_{j+1} \end{cases}$$

Luego, la función *b* es una de *Función de cuantificación con histéresis*. El ancho de la ventana de histéresis es e y los parámetros d_0 y d_r son los valores de saturación inferior y superior (ver Figura 14).

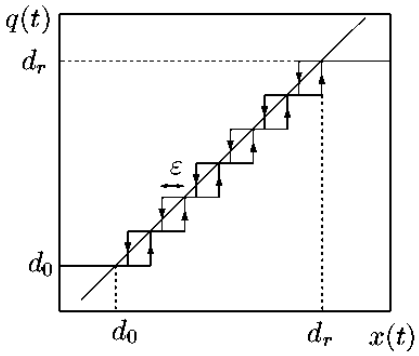


Figura 14. Función de Cuantificación con Histéresis

Consideremos ahora el siguiente sistema de ecuaciones de estado (SES):

$$\begin{cases} \dot{x} = f(x(t), u(t)) \\ y(t) = g(x(t), u(t)) \end{cases}$$

Asociado a este SES, definimos entonces el QSS de acuerdo a:

$$\begin{cases} \dot{x} = f(q(t), u(t)) \\ y(t) = g(q(t), u(t)) \end{cases}$$

donde q y x están relacionadas componente a componente por funciones de cuantificación con histéresis. Los componentes de q se llaman *variables cuantificadas*. La figura 15 muestra un Diagrama de Bloques de un QSS genérico.

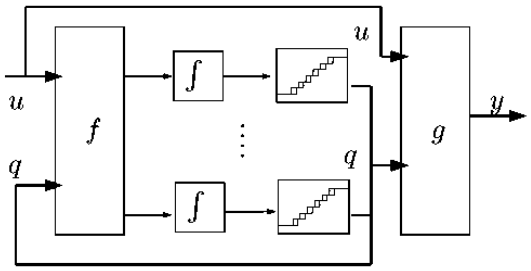


Figura 15. Diagrama de Bloques de un QSS.

Si la función f es acotada en cualquier dominio acotado, cuando la entrada u es seccionalmente constante puede asegurarse que:

- Las variables cuantificadas tienen trayectorias seccionalmente constantes
- Las variables de estado tienen trayectorias seccionalmente lineales
- Las derivadas de las variables de estado son seccionalmente constantes.

La demostración de estas propiedades puede encontrarse en (Kofman y Junco, 2001a). La Figura 16 muestra trayectorias típicas en un QSS. Esta forma particular de las trayectorias nos permitirá representar mediante un DEVS un QSS genérico a través de la representación de los cambios en las trayectorias mediante eventos.

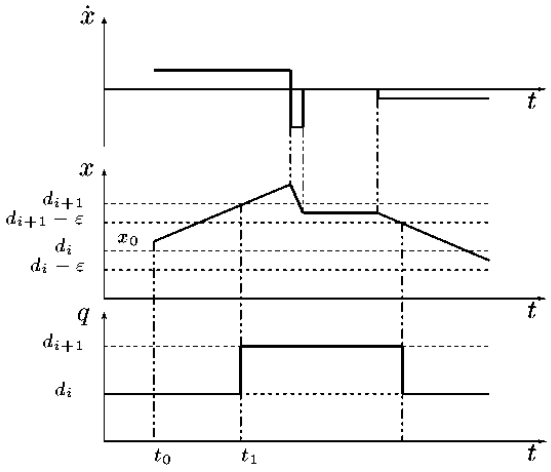


Figura 16. Trayectorias Típicas en QSS

8.2. Representación DEVS de un QSS.

Aprovechando las propiedades de acoplamiento modular de DEVS, dividiremos el QSS como muestra la figura 17.

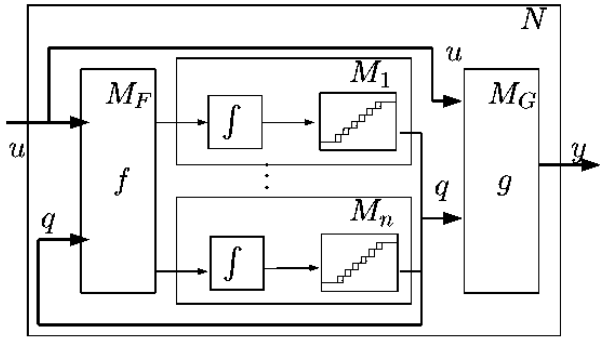


Figura 17. División en submodelos de un QSS

Los modelos M_1, ...,M_n corresponden a integradores con cuantificadores (o integradores cuantificados), en tanto que M_F y M_G corresponden a las funciones estáticas (cuya representación es casi trivial en términos de DEVS).

El modelo DEVS de un integrador cuantificado puede escribirse como sigue:

$$M_j =< X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta >, \text{ where:}$$
$$X = Y = \mathbb{R}$$
$$S = \mathbb{R} \times \mathbb{R} \times \mathbb{Z} \times \mathbb{R}_0^+ \infty$$
$$\delta_{\text{int}}(x, u, i, \sigma) = (x + \sigma u, u, i + sgn(u), \sigma')$$
$$\delta_{\text{ext}}(x, u, i, \sigma, e, v) = (x + eu, v, i, \sigma'')$$
$$\lambda(x, u, i, \sigma) = d_{i+sgn(u)}$$
$$ta(x, u, i, \sigma) = \sigma$$

donde

$$\sigma' = \begin{cases} \frac{d_{i+2} - (x + \sigma u)}{u} & if\ u > 0 \\ \frac{(x + \sigma u) - (d_{i-1} - \varepsilon)}{|u|} & if\ u < 0 \end{cases} \qquad \sigma'' = \begin{cases} \frac{d_{i+1} - (x + eu)}{v} & if\ v > 0 \\ \frac{(x + eu) - (d_i - \varepsilon)}{|v|} & if\ v < 0 \\ \infty & if\ v = 0 \end{cases}$$

Ejemplo 28: (ejercicio) Obtener el modelo DEVS correspondiente a una función estática genérica de una variable (*f* ó *g*).

Ejemplo 29: (ejercicio). Obtener la expresión del acoplamiento (mediante puertos) entre los distintos submodelos correspondientes a un QSS genérico.

Con el modelo del integrador cuantificado y los modelos de los ejercicios anteriores, tenemos todas las herramientas para simular un sistema de ecuaciones de estado genérico mediante DEVS. Sin embargo, resta decidir que cuantificación se debe utilizar en las diferentes variables de estado. Para eso, tendremos en cuenta ciertas propiedades de estabilidad y convergencia del método.

8.3. Estabilidad y Convergencia de QSS.

Aunque hasta aquí vimos que un QSS puede simularse por DEVS, esto no nos garantiza que QSS constituya una buena aproximación al SES original. Para esto, se demostraron las siguientes propiedades:

- Si el SES tiene un punto de equilibrio asintóticamente estable, entonces se puede encontrar una cuantificación que asegura que las trayectorias del QSS resultante converjan hacia regiones pequeñas en torno a dicho punto. (estabilidad)
- Si la función *f* satisface ciertas condiciones, las trayectorias del QSS convergen hacia las trayectorias del SES cuando la cuantificación tiende a cero (convergencia del método).

La primer propiedad nos permite hallar la cuantificación que asegura una cota en el error al final de la simulación. La segunda, dice que un error arbitrariamente pequeño puede ser alcanzado a lo largo de toda la simulación si se toman valores de cuantificación lo suficientemente pequeños.

Estas propiedades, cuyas demostraciones se encuentran en (Kofman y Junco, 2001a), implican que la aproximación de un SES por un QSS es un método de simulación bien condicionado, con las mismas (y en algún aspecto mejores) propiedades que los métodos clásicos de tiempo discreto.

En el caso particular de sistemas lineales y estacionarios, es posible encontrar además una fórmula cerrada que vincula la cuantificación con la cota de error a lo largo de toda la simulación (Kofman, 2001a).

Con respecto a la elección de la ventana de histéresis, en general la mejor opción es tomarla igual que un intervalo de cuantificación. Este aspecto es discutido en (Kofman et al., 2001a).

9. Comentarios finales.

El formalismo DEVS constituye una de las herramientas mas útiles recientemente desarrolladas para problemas generales de simulación de toda clase de sistemas. Pese a que en este apunte se intentó brindar un panorama bastante amplio, hay muchísimas cosas que quedaron fuera. Creo que es imprescindible mencionar las aplicaciones en sistemas distribuidos (ver Zeigler et. al, 2000) y de tiempo real. Entre estas últimas, hay una aplicación muy reciente en el control de sistemas continuos (Kofman, 2001b) que abre toda una serie de nuevos problemas que van desde la teoría de control hasta la simulación en tiempo real.

Lo reciente de su desarrollo hace que DEVS sea una fuente muy abundante de problemas abiertos, tanto en lo teórico como en cuestiones prácticas, abarcando tanto cuestiones de la matemática como de diferentes aspectos de las ciencias de la computación.

Referencias

Baccelli, F.; G.Cohen, G.Olsder and J.P.Quadrat (1992). *Synchronization and Linearity*. J.Wiley&Sons.

Cassandras, Christos (1993). *Discrete Event Systems: Modeling and Performance Analysis*. Irwin, Inc and Aksen Associates.

Giambiasi, N., Escude, B., and Ghosh, S. (2000). *GDEVS: A generalized Discrete Event specification for accurate modeling of dynamic systems*. Transactions of SCS, Vol. 17 No. 3, pp. 120-134.

Kofman, E. (2000). *Simulación de sistemas dinámicos por cuantificación de estados*. In Proceedings of AADECA 2000, pp. 435-430. Buenos Aires, Argentina.

Kofman, E. (2001a). *Algunas propiedades de la simulación de ODE's por cuantificación de estados*. In Proceedings of RPIC'01. Santa Fe, Argentina.

Kofman, E. (2001b). *Quantized-State Control. A Method for Discrete Event Control of Continuous Systems*. Accepted for publication in *Latin American Applied Research*, 2001.

Kofman E. and Junco, S. (2001a). *Quantized State Systems. A DEVS approach for continuous system simulation*. Transactions of SCS (to appear, sept. 2001).
(Available at www.eie.fceia.unr.edu.ar/~lsd/)

Kofman E. and Junco S. (2001b). *Quantized Bond Graphs: An Approach for Discrete Event Simulation of Physical Systems*. In *Proceedings of ICBGM'01* .pp. 369-374. Phoenix, Arizona.

Kofman, E., Lee, J.S. and Zeigler, B. (2001). *DEVS Representation of Differential Equation Systems: Review of Recent Advances.* In Proceedings of *ESS2001*.(to appear, oct. 2001)

Lewerentz, C.; T.Lindner y colaboradores (1995). *Formal Development of Reactive Systems*. Springer-Verlag.

Naamane, A., Damiba, A. and Giambiasi, N. (2001). *Bond Graphs and GDEVS*. In Proceedings of ICBGM'01, (Phoenix), pp. 375-380.

Zeigler, B., H.Sang Song, T.G.Kim and H.Praehofer (1995). *DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems*. Lecture Notes in Computer Science. Vol. 999, Springer. pp. 529-551.

Zeigler, B. and S. Vahie (1993). *DEVS Formalism and Methodology: Unity of Conception/Diversity of Application*. Proceedings of the Winter Simulation Conference, Washington D.C..

Zeigler, B. and Lee, J. S. (1998). *Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment*. SPIE Proceedings. Vol. 3369, pp. 49-58, 1998

Zeigler, B., H. Praehofer and T.G. Kim (2000). *Theory of Modeling and Simulation second ed*., Academic Press, New York.