Introducción a la Modelización y Simulación de Sistemas de Eventos Discretos con el Formalismo DEVS

Ernesto Kofman

Laboratorio de Sistemas Dinámicos
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

Índice General

Capítulo 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 [2]. 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 [1]. 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, Bernard Zeigler propuso a mediados de la década del '70 un formalismo general para la representación de dichos sistemas. Este formalismo, denominado DEVS (Discrete EVent System specification), constituye el formalismo más general para

el tratamiento de DES [10]. 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) [7].

Debido en parte a esto último, a la gran adaptación del formalismo para modelizar sistemas complejos y a la simplicidad y eficiencia de la implementación de simulaciones, DEVS es hoy en día una de las herramientas más utilizadas en modelado y simulación por eventos discretos.

Las aplicaciones del formalismo en este momento trascienden incluso la simulación de sistemas de eventos discretos, ya que se han desarrollado recientemente toda una gama de nuevas técnicas que permiten utilizar DEVS para aproximar ecuaciones diferenciales y simular sistemas continuos e híbridos [5]

En este apunte desarrollaremos varios de los principales aspectos del formalismo DEVS, basándonos principalmente en la segunda edición del libro *Theory of Modeling and Simulation* [8]. En lo subsiguiente, salvo que se cite a otra fuente, estaremos en general refiriéndonos a dicho libro.

Capítulo 2

Conceptos Previos

DEVS está basado en la teoría general de sistemas. Si bien no desarrollaremos aquí un curso sobre teoría de sistemas, intentaremos al menos introducir los principales conceptos que aplicaremos posteriormente al definir el formalismo DEVS.

2.1 Clasificación de Sistemas

Cuando hablamos de Sistemas de Eventos Discretos estamos asumiendo implícitamente que existe una categorización de los sistemas. El diferenciar los DES de los Sistemas Continuos o de los Sistemas de Tiempo Discreto conlleva una clasificación intrínseca en función de la naturaleza de los sistemas y de su lenguaje descriptivo.

Sin embargo esta no es la única forma de clasificar los sistemas. Hay una manera transversal que tiene que ver con el nivel de especificación de la descripción y que se puede aplicar a sistemas de distinta naturaleza. Un sistema puede estar especificado de una forma muy vaga, conociéndose símplemente quienes son sus entradas y salidas, o bien puede estar descripto de una manera mucho más detallada, con información sobre la estructura y el funcionamiento de cada uno de sus componentes.

En síntesis, podemos decir que la Teoría de Sistemas clasifica los sistemas según:

- Formalismo de especificación: Diferentes formas de modelado que conducen a especificaciones continuas o discretas, ya sean en el tiempo o en las variables descriptivas.
- Nivel de especificación: Diferentes niveles en los cuales es posible describir el comportamiento del sistema y los mecanismos que producen ese comportamiento.

En cuanto a los formalismos de especificación de sistemas, encontramos básicamente tres grandes clases, clasificadas de acuerdo a la forma de cam-

bio 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.

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 Tabla 2.1 muestra un resumen de la misma:

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.

Como dijimos antes, en este apunte trataremos casi exclusivamente con Sistemas de Eventos Discretos en lo que refiere a la naturaleza, utilizando el formalismo DEVS. Con respecto a los niveles de especificación trabajaremos principalmente con los dos más altos (Transición de Estados y Componentes Acoplados), aunque también trataremos habitualmente con descripciones (en general verbales) correspondientes a los niveles inferiores.

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	Cómo los estados son afectados por las entradas y como las salidas son afectados por la entrada y el estado actual.
4	Componentes acoplados	Qué componentes tiene el sistema y cómo se acoplan entre sí. Los componentes pueden estar dados a bajo nivel o pueden ser nuevas estructuras.

Tabla 2.1: Niveles de Especificación de un Sistema

2.2 Especificación de Comportamiento de un Sistema Dinámico

La Teoría de Sistemas utiliza habitualmente estructuras formadas por conjuntos y funciones para especificar los sistemas.

Existe en particular una estructura capaz de representar el comportamiento de cualquier sistema dinámico determinista, independientemente de su característica continua, discreta o híbrida.

Formalmente, cualquier especificación de comportamiento de un sistema puede ser representada por la siguiente estructura:

$$S_D = (T, X, Y, \Omega, Q, \Delta, \Lambda), \text{ donde}$$

- T es la *base de tiempo*.
- X es el conjunto de valores que puede tomar la entrada.
- Y es el conjunto de valores que puede tomar la salida.
- Ω es el conjunto de los segmentos de entrada admisibles: $\Omega : \{\omega : (t_1, t_2) \rightarrow X\}$ siendo (t_1, t_2) un intervalo en T.
- Q es el conjunto de valores de los estados.
- $\Delta : Q \times \Omega \rightarrow Q$ es la función de transición global.

- $\Lambda : Q \times X \to Y$ es la función de salida.

La interpretación puede ser la siguiente: dado un segmento de trayectoria de entrada ω (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 un instante, en tanto, es simplemente un reflejo del valor estado y de la entrada en dicho instante (a través de la función de salida por supuesto).

Esto es, supongamos que en un instante $t_0 \in T$ el estado vale $q_0 \in Q$ y en el intervalo (t_0, t_1) (con $t_1 \in T$) la entrada está dada por el segmento $\omega \in \Omega$. Luego, en el tiempo t_1 se tendrá que el estado vale $q_1 = \Delta(q_0, \omega(t) \in Q)$. Asímismo, la salida en el instante t_1 estará dada por $y_1 = \Lambda(q_1, w(t1))$.

Capítulo 3

El Formalismo DEVS

Como se mencionó en la introducción, el formalismo DEVS fue concebido como una herramienta general de modelización y simulación de *Sistemas de Eventos Discretos*.

DEVS permite representar cualquier sistema que experimente un número finito de cambios (eventos) en cualquier intervalo de tiempo. De esta forma, podremos ver que DEVS es en realidad un caso particular de la representación general de sistemas dinámicos vista en la página 6, en la cual las trayectorias de entrada estarán restringidas a *segmentos de eventos* y la función de transición tendrá una forma especial que limitará las trayectorias de salida para que tengan idéntica naturaleza.

Un *segmento de eventos* se define formalmente de la siguiente manera:

Definición 3.1. *Segmento de Eventos.*

Sea $\omega : [t_0, t_n] \rightarrow A \cup \{\phi\}$ *un segmento sobre una base continua de tiempo (o sea, una función de* $t \in (t_0, t_n)$*, siendo este último un intervalo de los reales) y el conjunto* $A \cup \{\phi\}$*. Aquí* ϕ *es un elemento que no pertenece a* A *y representa la ausencia de evento.*

Luego, ω *es un segmento 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)*.*

La Figura 3.1 ilustra una trayectoria de eventos, en la cual los valores a_i pertenecen al conjunto A.

Más allá de esta definición formal, un *evento* es la representación de un cambio instantáneo en alguna parte de un sistema. El mismo puede caracterizarse por un valor y un instante en el que ocurre. El valor puede ser un número, un vector, una palabra o, en general, un elemento cualquiera de un conjunto determinado (A en nuestra definición).

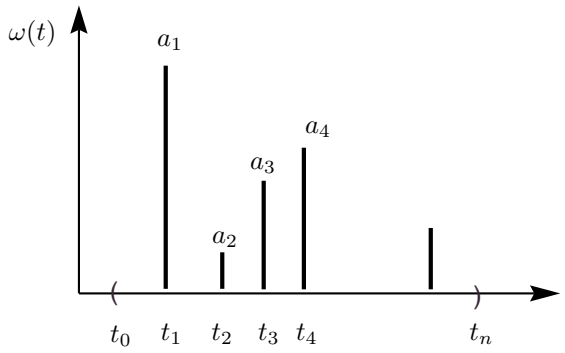

Figura 3.1: Trayectoria de Eventos

3.1 Modelos DEVS Atómicos

Un modelo DEVS procesa una trayectoria de eventos de entrada y, según esta trayectoria y sus propias condiciones iniciales, produce una trayectoria de eventos de salida. Este comportamiento entrada/salida se representa en la Figura 3.2.

Figura 3.2: Comportamiento Entrada/Salida de un modelo DEVS

Un modelo DEVS *atómico* queda definido por la siguiente estructura:

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

donde:

- X es el conjunto de valores de eventos de entrada, es decir el conjunto de todos los valores que un evento de entrada puede adoptar.
- Y es el conjunto de valores de eventos de salida.
- S es el conjunto de valores de estado.
- δ_{int}, δ_{ext}, λ y ta son funciones que definen la dinámica del sistema.

Cada posible estado s ($s \in S$) tiene asociado un *Avance de Tiempo* calculado por la *Función de Avance de Tiempo* (Time Advance Function) $ta(s)$ ($ta(s)$:

$S \to \mathbb{R}_0^+$). El *Avance de Tiempo* es un número real no negativo que indica cuanto tiempo el sistema permanecerá en un estado determinado en ausencia de eventos de entrada.

Luego, si el estado toma el valor s_1 en el tiempo t_1, tras $ta(s_1)$ unidades de tiempo (o sea, en tiempo $ta(s_1) + t_1$) el sistema realiza una *transición interna* yendo a un nuevo estado s_2. El nuevo estado se calcula como $s_2 = \delta_{\text{int}}(s_1)$. La función δ_{int} ($\delta_{\text{int}} : S \to S$) se llama *Función de Transición Interna* (*Internal Transition Function*).

Cuando el estado va de s_1 a s_2 se produce también un evento de salida con valor $y_1 = \lambda(s_1)$. La función λ ($\lambda : S \to Y$) se llama *Función de Salida* (*Output Function*). Así, las funciones ta, δ_{int} y λ definen el comportamiento autónomo de un modelo DEVS.

Cuando llega un evento de entrada el estado cambia instantáneamente. El nuevo valor del estado depende no sólo del valor del evento de entrada sino también del valor anterior de estado y del tiempo transcurrido desde la última transición. Si el sistema llega al estado s_3 en el instante t_3 y luego llega un evento de entrada en el instante $t_3 + e$ con un valor x_1, el nuevo estado se calcula como $s_4 = \delta_{\text{ext}}(s_3, e, x_1)$ (notar que $ta(s_3) > e$). En este caso se dice que el sistema realiza una *transición externa*. La función δ_{ext} ($\delta_{\text{ext}} : S \times \mathbb{R}_0^+ \times X \to S$) se llama *Función de Transición Externa* (*External Transition Function*). Durante una transición externa no se produce ningún evento de salida.

La Figura 3.3 ilustra trayectorias típicas de un modelo DEVS. En la misma puede observarse que el sistema está originalmente en un estado s_1.

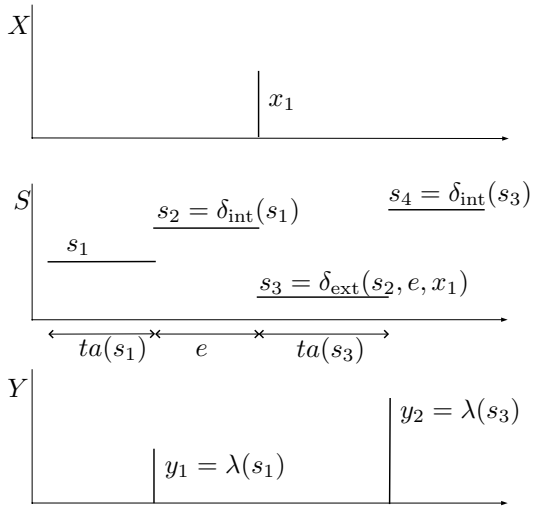

Figura 3.3: Comportamiento de un modelo DEVS

Transcurrido el tiempo $ta(s_1)$, dado que no hubo ningún evento de entrada, el sistema realiza una transición interna. Tras esta transición el estado toma

el valor $s_2 = \delta_{\text{int}}(s1)$. En el mismo instante, se produce un evento de salida $y_1 = \lambda(s_1)$.

Luego, antes que transcurra el nuevo tiempo de avance $ta(s_2)$ el sistema recibe un evento de entrada con valor x_1. Entonces, se produce una transición externa y el estado adopta el nuevo valor $s_3 = \delta_{\text{ext}}(s_1, e, x_1)$ siendo e el tiempo transcurrido desde el evento anterior (cuando el estado pasó a s_2) hasta el momento en que llega el evento.

Transcurrido el tiempo $t_a(s_3)$ y sin que haya habido ningún evento de entrada en el interín, el sistema realiza una nueva transición interna yendo al estado $s_4 = \delta_{\text{int}}(s_3)$ y provocando el evento de salida $y_2 = \lambda(s_3)$.

Tras el ejemplo de la Figura 3.3 debería quedar claro cómo puede analizarse el comportamiento de un modelo DEVS atómico. Veremos entonces ahora algunos ejemplos en los cuales partiendo de la especificación del comportamiento obtendremos modelos DEVS que respondan a la misma.

Ejemplo 3.1. *Procesador Elemental*

Consideremos un sistema que recibe números positivos de manera asincrónica. Cuando recibe un número x, tras $3 \cdot x$ unidades de tiempo produce un evento con el número $x/2$.

Puede verse fácilmente que el siguiente modelo DEVS verifica este comportamiento:

$$
\begin{aligned}
&M_1 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \textit{ donde} \\
&X = Y = S = \Re^+ \\
&\delta_{int}(s) = \infty \\
&\delta_{ext}(s, e, x) = x \\
&\lambda(s) = s/2 \\
&ta(s) = 3 \cdot s
\end{aligned}
$$

Observar que en el modelo DEVS M_1 el estado puede tener un avance de tiempo igual a ∞. Cuando esto ocurre, decimos que el sistema está en un estado *pasivo*, ya que no hará nada hasta que se reciba un nuevo evento.

Veamos entonces que ocurre con el sistema M_1 cuando recibe una trayectoria de eventos. Consideremos por ejemplo que hay eventos cuando $t = 1$, $t = 3$ y $t = 10$ con valores 2, 1 y 5 respectivamente. Supongamos que inicialmente tenemos $t = 0$, $s = \infty$ y $e = 0$. Luego, el comportamiento observado sería el siguiente:

tiempo $t = 0$:
- $s = \infty$
- $e = 0$
- $ta(s) = ta(\infty) = \infty$

tiempo $t = 1^-$:
- $s = \infty$
- $e = 1$

tiempo $t = 1$:
$s = \delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}(\infty, 1, 2) = 2$

tiempo $t = 1^+$:
$s = 2$
$e = 0$
$ta(s) = ta(2) = 6$

tiempo $t = 3^-$:
$s = 2$
$e = 2$

tiempo $t = 3$:
$s = \delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}(2, 2, 1) = 1$

tiempo $t = 3^+$:
$s = 1$
$e = 0$
$ta(s) = ta(1) = 3$

tiempo $t = 6$:
evento de salida con valor $\lambda(s) = \lambda(1) = 0.5$
$s = \delta_{\text{int}}(s) = \delta_{\text{int}}(1) = \infty$

tiempo $t = 6^+$:
$s = \infty$
$e = 0$
$ta(s) = ta(\infty) = \infty$

tiempo $t = 10^-$:
$s = \infty$
$e = 4$

tiempo $t = 10$:
$s = \delta_{\text{ext}}(s, e, x) = \delta_{\text{ext}}(\infty, 4, 5) = 5$

tiempo $t = 10^+$:
$s = 5$
$e = 0$
$ta(s) = ta(5) = 15$

tiempo $t = 25$:
evento de salida con valor $\lambda(s) = \lambda(5) = 2.5$
$s = \delta_{\text{int}}(s) = \delta_{\text{int}}(5) = \infty$

tiempo $t = 25^+$:
$s = \infty$
$e = 0$
$ta(s) = ta(\infty) = \infty$

De acuerdo al modelo anterior, cuando un número nuevo llega antes que el anterior haya producido el evento el sistema toma el nuevo y se olvida del otro (esto es lo que pasa cuando $t = 3$).

Un sistema similar, un poco más general y en el cual no ocurre lo anterior es el del siguiente ejemplo:

Ejemplo 3.2. *Modelo de un Procesador*

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 dado por una función $t_p(\cdot)$. Una vez transcurrido este tiempo, el procesador provoca un evento en el que muestra el "resultado"del trabajo (una función $y : J \to J$ conocida, donde J es el conjunto de trabajos)[1]*.*

Si cuando llega un evento el procesador estaba procesando un trabajo, este nuevo evento debe ser ignorado. Un modelo DEVS de este sistema puede ser el que sigue:

$$
\begin{aligned}
&M_2 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \textit{ donde} \\
&X = Y = J \triangleq \{job_1, job_2, \cdots\} \\
&S = J \cup \{\phi\} \times \Re_0^+ \\
&\delta_{int}(s) = \delta_{int}((job, \sigma)) = (\phi, \infty) \\
&\delta_{ext}(s, e, x) = \delta_{ext}((job, \sigma), e, x) = \begin{cases} (x, t_p(x)) & \textit{si } job = \phi \\ (job, \sigma - e) & \textit{en otro caso} \end{cases} \\
&\lambda(s) = \lambda((job, \sigma)) = y(job) \\
&ta(s) = ta((job, \sigma)) = \sigma
\end{aligned}
$$

En este último modelo puede observarse que en el estado s, además del último trabajo, se guarda el tiempo que falta para que finalice el mismo (σ). Esto es necesario ya que cuando llega un evento que debe ignorarse (cuando el procesador está ocupado) el tiempo restante (definido en la función $ta(s)$) estará determinado en función del tiempo transcurrido e. Por esto, conocer sólo el trabajo que se está procesando no es suficiente para determinar el tiempo de avance.

Utilizar una variable σ igual al tiempo de avance es muy común en DEVS (de hecho, es lo que haremos en todos los modelos de aquí en adelante). De esta forma, para ignorar un evento de entrada se debe dejar el resto del estado como estaba, restándole e al valor que tenía σ.

Ejemplo 3.3. *Modelo de una cola.*

Consideremos 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*

[1] Como puede verse, no estamos teniendo en cuenta lo que internamente realiza el procesador. Sólo consideramos el tiempo que lleva hacerlo (y lo suponemos conocido).

(y descargar) el primer trabajo de dicha cola. La transmisión de dicho trabajo se realiza mediante un evento de salida con dicho trabajo. Supondremos además que la cola demora un tiempo nulo en provocar la salida.

Un modelo del sistema descripto está dado por la siguiente estructura[2]*:*

$$M_3 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta),\ donde$$
$$X = J \cup \{\text{“}ready''\}$$
$$Y = J$$
$$S = J^+ \cup \{\phi\} \times \Re_0^+$$
$$\delta_{int}(s) = \delta_{int}((q \bullet job, \sigma)) = (q, \infty)$$
$$\delta_{ext}(s, e, x) = \delta_{ext}((q, \sigma), e, x) = \begin{cases} (q, 0) & si\ x = \text{“}ready\text{”} \\ (x \bullet q, \infty) & en\ otro\ caso \end{cases}$$
$$\lambda(s) = \lambda((q \bullet job, \sigma)) = job$$
$$ta(s) = ta((q, \sigma)) = \sigma$$

En este nuevo ejemplo hay una particularidad: tras la llegada del evento “ready” se debe provocar un evento de salida de manera inmediata. Como no se producen eventos de salida durante las transiciones externas hay que realizar un pequeño truco: colocar el avance de tiempo en 0 y forzar una transición interna inmediatamente. Este estado se denomina estado *transitorio* ya que el sistema sólo estará en el mismo durante un instante.

3.2 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 producirs). 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”.

Si bien *Parallel DEVS* tiene ventajas respecto al enfoque DEVS clásico presentado aquí, no será tratado aquí por exceder los objetivos de este curso. Quien tenga interés en este nuevo formalismo puede consultar la principal referencia de este apunte [8].

[2]El conjunto J es el mismo del ejemplo anterior, mientras que J^+ es la versión extendida de J considerando todas las secuencias de elementos de J

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 formalismo 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:

$$S_D = (T, X, Y, \Omega, Q, \Delta, \Lambda), \text{ donde}$$

Para que una estructura definida por el modelo DEVS:

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

defina realmente un Sistema Dinámico, esta estructura debería ser un caso particular de la anterior. Veremos entonces (de manera constructiva) como se relacionan las mismas:

- La base de tiempo T de la estructura S_D es el conjunto de los números reales $\Re$
- El conjunto de valores de entrada está dado por $X = X' \cup \{\phi\}$, o sea, el conjunto de valores de entrada del sistema DEVS junto al símbolo ϕ que denota la ausencia de evento.
- El conjunto de valores de salida está dado por $Y = Y' \cup \{\phi\}$.
- El conjunto de estados Q es el conjunto de estado total Q' de DEVS, siendo este el conjunto de pares ordenados (s, e) con $s \in S$ y $e \in \Re^+$.
- El conjunto Ω de segmentos de entrada admisible es el conjunto de los segmentos de eventos con valores en X'.

Con esto sólo resta definir las funciones de transición y de salida. Observamos entonces también que:

- Las trayectorias de estado son seccionalmente constantes sobre S y T.
- Las trayectorias de salidas son segmentos de eventos sobre Y y T.

En base a esto, la función de transición Δ queda definida como sigue:

Sea $\omega : (t_i, t_f] \to X$ un segmento de eventos y sea el estado $q = (s, e)$ el estado del sistema en tiempo t_i. Entonces:

$$\Delta(q, \omega) = \begin{cases} (s, e + t_f - t_i) & \text{si } e + t_f - t_i < ta(s) \wedge \\ & \omega(t) = \phi \quad \forall t \in (t_i, t_f) \\ (\Delta(\delta_{\text{int}}(s), 0, \omega')) & \text{si } e + t_f - t_i \geq ta(s) \wedge \\ & \omega(t) = \phi \quad \forall t \in (t_i, t_i + ta(s) - e) \\ \Delta((\delta_{\text{ext}}((s, e + t_1 - ti), & \\ \omega(t_1)), 0), \omega'') & \text{en otro caso} \end{cases}$$

Donde ω' es un segmento igual a ω pero definido sólo en el intervalo $[t_i + ta(s) - e, t_f)$. Similarmente, ω'' es un segmento igual a ω pero definido en el intervalo $[t_1, t_f)$, donde además se cumple que $\omega''(t_1) = \phi$. El valor t_1 es el menor valor en el intervalo $[t_i, t_f)$ para el cual $\omega(t_1) \neq \phi$ (tiempo del primer evento).

Como puede verse, en el primer caso consideramos que no ocurre ningún evento (interno ni externo), en el segundo consideramos que ocurre primero un evento interno mientras que en el último consideramos que ocurre primero un evento externo (en tiempo t_1).

Finalmente, la función de salida Λ está dada por:

$$\Lambda((s,e),x) = \begin{cases} \lambda(s) & \text{si } e = ta(s) \wedge x = \phi \\ \lambda(s) \text{ ó } \phi & \text{si } e = ta(s) \wedge x \neq \phi \\ \phi & \text{en otro caso} \end{cases}$$

donde la segunda opción depende de que se considere prioritario (transiciones internas o externas).

Resta 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 los Modelos 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, deberemos asegurar que haya un número finito de eventos en el intervalo (t_i, t_f). Dado que el número de eventos externos será siempre finito (por la definición de ω como segmento de eventos) bastará con garantizar que el total de transiciones internas sea finito.

De esta forma, la condición para que un modelo DEVS defina un sistema dinámico (o sea, para que sea legitimado) dependerá exclusivamente de la función de transición interna y del tiempo de avance.

Para establecer entonces las condiciones suficientes y necesarias de legitimidad comenzaremos definiendo una función de transición interna extendida $\delta^+_{\text{int}} : S \times \mathbb{N}_0 \to S$, con la siguiente recursión:

$$\begin{aligned} \delta^+_{\text{int}}(s,0) &= s \\ \delta^+_{\text{int}}(s,n+1) &= \delta_{\text{int}}(\delta^+_{\text{int}}(s,n)) \end{aligned}$$

A partir de esta, y de manera similar, definimos la función $\Sigma : S \times \mathbb{N}_0 \to \Re_0^\infty$ como:

$$\begin{aligned} \Sigma(s,0) &= 0 \\ \Sigma(s,n) &= \Sigma(s,n-1) + ta(\delta_{\text{int}}^+(s,n-1)) \end{aligned}$$

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

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

En este caso, se dice que el sistema DEVS es legitimado. La interpretación es sencilla: puede verse fácilmente que $\Sigma(s,n)$ es el tiempo total transcurrido desde que el sistema llega al estado s hasta que realiza $n-1$ transiciones. (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 infinito de eventos en un intervalo de tiempo finito.

Cuando existe un ciclo cerrado en el diagrama de estados de δ_{int} (o sea, una sucesión cíclica de estados producto de transiciones internas sucesivas) que contiene sólamente 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 la paradoja 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 más 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.

3.5 Problemas Propuestos

[P3.1] Modelo de un Generador de Eventos.

Obtener un modelo DEVS de un sistema que produzca eventos cada t_s unidades de tiempo con un valor cualquiera.

[P3.2] Retardos en la Cola.

Modificar el Ejemplo 3.3 para representar un retardo entre la llegada de la señal de "ready" y la transmisión del trabajo correspondiente.

[P3.3] Cola Revisitada.

En el Ejemplo 3.3, 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").

[P3.4] Cola con Almacenamiento Limitado.

Modificar el Problema [P3.3] 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.

[P3.5] Función Estática

Un sistema recibe eventos con números reales de manera asincrónica (es decir, puede recibirlos en cualquier momento). Cada vez que el sistema recibe un evento produce otro inmediatamente con el valor $f(x)$ siendo x el número recibido y f una función conocida.

Obtener un modelo DEVS del sistema descripto.

[P3.6] Integrador I

Un sistema recibe eventos con números reales que representan una trayectoria seccionalmente constante (cada vez que la trayectoria cambia, se recibe un evento con el nuevo valor). El sistema calcula la integral de la trayectoria mencionada y cada una unidad de tiempo (o cada un segundo si se quiere) provoca un evento con el valor de dicha integral.

Esto es, si el sistema recibe eventos con valores 2 y 5 en los instantes $t = 0$ y $t = 4$, comenzará enviando un evento con el valor 0 en $t = 0$, luego 2 en $t = 1$, luego 4 en $t = 2$, 6 en $t = 3$, 8 en $t = 4$, 13 en $t = 5$, 18 en $t = 6$, etc.

Modelizar este sistema utilizando DEVS.

[P3.7] Legitimidad I

Demostrar la legitimidad de todos los modelos vistos en los ejemplos y problemas de este capítulo.

[P3.8] Legitimidad II

Considerar el siguiente modelo DEVS:

$$
\begin{aligned}
&M_4 = (X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \text{ donde} \\
&X = Y = S = \Re^+ \\
&\delta_{\text{int}}(s) = s/2 \\
&\delta_{\text{ext}}(s, e, x) = x \\
&\lambda(s) = s \\
&ta(s) = s
\end{aligned}
$$

Determinar si es o no legitimado.

Capítulo 4

Modelos DEVS Acoplados

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 más fácil es describir el comportamiento de varios componentes elementales y luego especificar como interactúan entre sí.

En general (y particularmente en DEVS) hay 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.

Dejando de lado los autómatas celulares y algunos otros casos especiales, en general es bastante más simple y adecuado el trabajo mediante acoplamiento modular ya que puede aislarse y analizarse cada componente independientemente del contexto en que se encuentre. Por este motivo, nos concentraremos exclusivamente en este tipo de acoplamiento dentro del cual distinguiremos dos formas: Acoplamiento mediante interfases de traducción y acoplamiento mediante puertos, siendo esta última en realidad un caso particular de la primera.

4.1 Acoplamiento Modular Básico

La Figura 4.1 muestra un modelo DEVS acoplado N. En la misma, se observan dos modelos *atómicos*, a y b y cinco *funciones de traducción*.

La función de traducción $Z_{a,b}$ transforma las salidas del modelo a en entradas del modelo b. Asumiendo que:

$$M_a = (X_a, Y_a, S_a, \delta_{\text{int}\,a}, \delta_{\text{ext}\,a}, \lambda_a, ta_a)$$

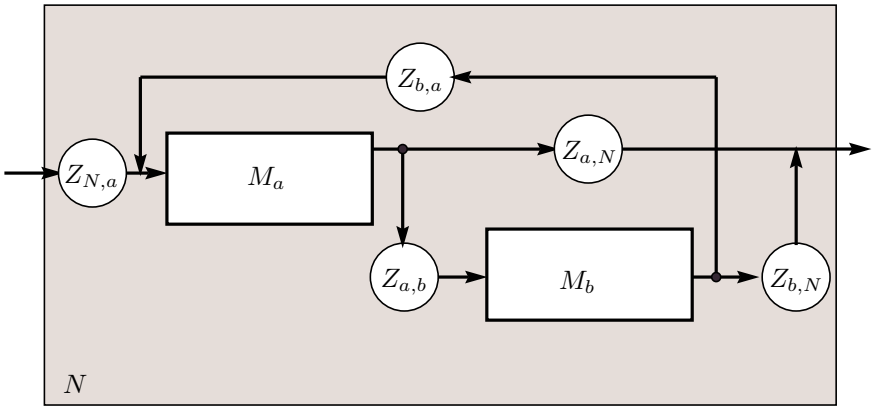

Figura 4.1: Modelo DEVS acoplado

y que

$$M_b = (X_b, Y_b, S_b, \delta_{\text{int}\,b}, \delta_{\text{ext}\,b}, \lambda_b, ta_b)$$

la función de traducción $Z_{a,b}$ deberá tener dominio en Y_a e imagen en X_b.

Si además llamamos X_N al conjunto de valores de entrada en el modelo acoplado N y llamamos Y_N al conjunto de valores de salida en N, el resto de las funciones de traducción serán:

$$\begin{aligned} Z_{b,a} &: Y_b \to X_a \\ Z_{N,a} &: X_N \to X_a \\ Z_{a,N} &: Y_a \to Y_N \\ Z_{b,N} &: Y_b \to Y_N \end{aligned}$$

Especificando todas estas funciones de traducción quedará entonces completamente determinado el funcionamiento del modelo N.

Formalmente, un modelo acoplado N cualquiera queda definido por la estructura:

$$N = (X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$$

donde:

- X_N es el conjunto de valores de entrada del sistema acoplado.
- Y_N 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.
- Para cada $d \in D \cup N$, I_d es el conjunto de modelos que influyen sobre el modelo d. De acuerdo a esto, deberá ser $I_d \subseteq D \cup N$ pero con la restricción

que $d \notin I_d$, esto es, un modelo no puede ser influyente de sí mismo, o sea, están prohibidas las conexiones desde un componente hacia si mismo así como una conexión directa desde la entrada hasta la salida del sistema acoplado.

- Para cada $i \in I_d$, $Z_{i,d}$ es la función de traducción donde:

$$\begin{aligned} Z_{i,d} &: X_N \to X_d \text{ si } i = N \\ Z_{i,d} &: Y_i \to Y_N \text{ si } d = N \\ Z_{i,d} &: Y_i \to X_d \text{ en otro caso} \end{aligned}$$

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

En el ejemplo de la Figura 4.1 ya dijimos como eran las funciones de traducción. Los conjuntos en tanto serán:

$$D = \{a, b\}$$
$$\{M_d\} = \{M_a, M_b\}$$
$$I_a = \{N, b\}, \qquad I_b = \{a\}, \qquad I_N = \{a, b\}$$

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*.

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 4.1. *Sistema Cola–Procesador.*

Supongamos que conectamos el procesador del Ejemplo 3.2 con la cola del Ejemplo 3.3 de forma tal que los trabajos lleguen a la cola y ésta los envíe al procesador a medida que el mismo se desocupe.

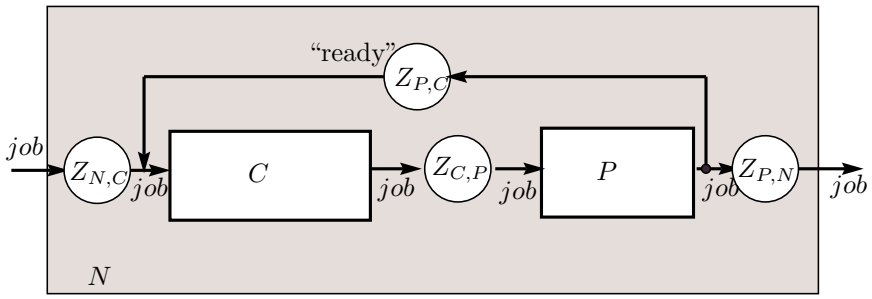

Figura 4.2: Modelo Acoplado Cola–Procesador

La Figura 4.2 muestra el esquema del sistema acoplado, que puede expresarse formalmente como:

$$
\begin{aligned}
&N = (X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select), \text{ donde} \\
&X_N = Y_N = J \triangleq \{job_1, job_2, \cdots\} \\
&D = \{C, P\} \\
&M_C = M_2, \;\; M_P = M_3 \\
&I_C = \{N, P\}, \;\; I_P = \{C\}, \;\; I_N = \{P\} \\
&Z_{N,C}(job) = job, \; Z_{P,C}(job) = \text{“ready”}, \\
&Z_{C,P}(job) = job, \; Z_{P,N}(job) = job \\
&Select(\{P, C\}) = C
\end{aligned}
$$

4.2 Clausura Bajo Acoplamiento DEVS

Una de las propiedades fundamentales del acoplamiento modular DEVS es la clausura. El cumplimiento de esta propiedad garantiza que el acoplamiento de modelos DEVS 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 *acoplamiento jerárquico*.

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 más general, puede utilizarse en el contexto de otro modelo acoplado sin necesidad de realizar modificaciones.

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

La estructura acoplada

$$N = (X_N, Y_N, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$$

de los modelos atómicos

$$M_d = (X_d, Y_d, S_d, \delta_{\text{int}\,d}, \delta_{\text{ext}\,d}, \lambda_d, ta_d)$$

define un modelo atómico:

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

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 en el sistema M_d hasta la última transición en el sistema N ($e_d \geq 0$)[1].

La función de transición externa del sistema acoplado es:

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

donde $s = (\cdots, (s_d, e_d), \cdots)$ y $\tilde{s} = (\cdots, (\tilde{s}_d, \tilde{e}_d), \cdots)$ con

$$(\tilde{s}_d, \tilde{e}_d) = \begin{cases} (\delta_{\text{ext}\,d}(s_d, e_d + e, x_d), 0) & \text{si } N \in I_d \wedge x_d \triangleq Z_{N,d}(x) \neq \phi \\ (s_d, e_d + e) & \text{en otro caso} \end{cases}$$

La función de avance de tiempo en tanto es:

$$ta(s) = \min_{d \in D} \sigma_d$$

siendo $\sigma_d \triangleq ta_d(s_d) - e_d$ el tiempo para la próxima transición interna del modelo M_d.

La función de transición interna queda definida por:

$$\delta_{\text{int}}(s, e, x) = \tilde{s}$$

y como antes $s = (\cdots, (s_d, e_d), \cdots)$ y $\tilde{s} = (\cdots, (\tilde{s}_d, \tilde{e}_d), \cdots)$.

Sea $IMM(s) \triangleq \{d | d \in D \wedge \sigma_d = ta(s)$ el conjunto de componentes *inminentes* (candidatos a realizar la próxima transición). Definimos entonces:

$$d^* = Select(IMM(s))$$

Es decir, M_{d^*} es el modelo que realizará la transición interna por ser el prioritario. Luego,

$$(\tilde{s}_d, \tilde{e}_d) = \begin{cases} (\delta_{\text{int}\,d}(s_d), 0) & \text{si } d = d^* \\ (\delta_{\text{ext}\,d}(s_d, e_d + ta(s), x_d), 0) & \text{si } d^* \in I_d \wedge x_d \triangleq Z_{d^*,d}(\lambda(s_{d^*})) \neq \phi \\ (s_d, e_d + ta(s)) & \text{en otro caso} \end{cases}$$

Por último, la función de salida es (bajo la misma definición de d^*)

$$\lambda(s) = \begin{cases} Z_{d^*,N}(\lambda(s_{d^*})) & \text{si } d^* \in I_N \\ \phi & \text{en otro caso} \end{cases}$$

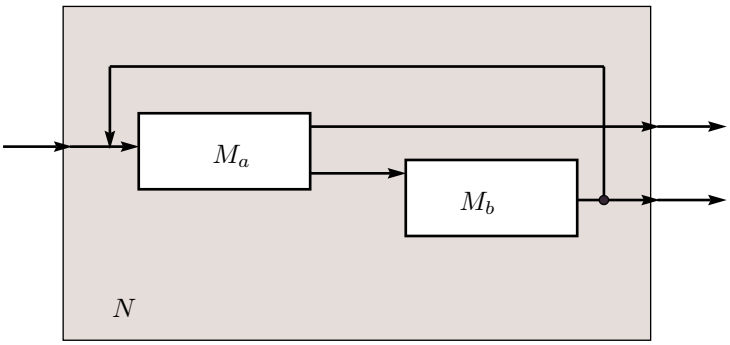

Figura 4.3: Modelo DEVS acoplado con puertos

4.3 Acoplamiento DEVS con Puertos

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.

Por un lado, la mayor parte de los dispositivos o procesos que se modelan por DEVS tienen una separación física o lógica intrínseca entre los diferentes tipos de entradas que pueden recibir o de salidas que pueden emitir. De esta forma, es natural representar que dichos eventos de entrada y de salida ocurren en diferentes puertos.

Por otro lado, es bastante engorroso trabajar con funciones de traducción ya que, salvo en el caso de dispositivos que contienen interfaces reales, su presencia es bastante artificial.

Por esto, la mayor parte de las herramientas de software de simulación con DEVS utilizan acoplamiento mediante puertos de entrada y salida reemplazando así el uso de las funciones de traducción.

En el formalismo DEVS, es posible introducir el concepto de puertos particularizando los conjuntos X e Y (conjuntos de valores de entrada y salida respectivamente) de los modelos atómicos.

De esta forma, puede verse también que el acoplamiento con puertos no es más que un caso especial del acoplamiento visto en la Sección 4.1 por lo que se cumplirá también la propiedad de clausura. Así, será posible también realizar acoplamiento jerárquico mediante puertos.

El conjunto de entrada de un modelo atómico con puertos deberá ser de la forma:

$$X = \{(p, v) | p \in InPorts,\ v \in X_p\}$$

siendo $InPorts$ el conjunto de puertos de entrada y X_p el conjunto de valores de entrada en el puerto p.

[1] Decimos que N tiene una transición cuando alguno de los subsistemas la tiene.

Similarmente, el conjunto de valores de salida será:

$$Y = \{(p, v) | p \in OutPorts,\ v \in Y_p\}$$

con definiciones análogas .

Ejemplo 4.2. *Cola con Puertos. Consideremos el Ejemplo 3.3. En dicho modelo podemos asumir que hay dos puertos de entrada. Por el primero llegan trabajos y por el otro la señal "ready".*

El modelo DEVS con puertos queda:

$$\begin{aligned}
&M_5 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta),\ \textit{donde}\\
&X = \{inport_1\} \times J \cup \{inport_2\} \times \{\text{“}ready''\}\\
&Y = \{outport_1\} \times J\\
&S = J^+ \cup \{\phi\} \times \Re_0^+\\
&\delta_{int}(s) = \delta_{int}((q \bullet job, \sigma)) = (q, \infty)\\
&\delta_{ext}((q, \sigma), e, (p, x_v)) = \begin{cases} (q, 0) & si\ p = inport_2\\ (x_v \bullet q, \infty) & en\ otro\ caso \end{cases}\\
&\lambda(s) = \lambda((q \bullet job, \sigma)) = (outport_1, job)\\
&ta(s) = ta((q, \sigma)) = \sigma
\end{aligned}$$

El acoplamiento de modelos con puertos en tanto se especifica mediante la siguiente estructura:

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

- El conjunto de valores de entrada del modelo acoplado es $X_N = \{(p, v) | p \in InPorts,\ v \in X_p\}$ siendo $InPorts$ el conjunto de puertos de entrada y X_p el conjunto de valores de entrada en el puerto p.
- El conjunto de valores de salida del modelo acoplado es $Y_N = \{(p, v)\},\ p \in OutPorts,\ v \in Y_p$ siendo $OutPorts$ el conjunto de puertos de entrada e Y_p el conjunto de valores de salida en el puerto p.
- D es el conjunto de referencias a componentes.
- Para cada $d \in D$, $M_d = (X_d, Y_d, S_d, \delta_{\text{int}\,d}, \delta_{\text{ext}\,d}, \lambda_d, ta_d)$ donde:
  - $X_d = \{(p, v) | p \in InPorts_d,\ v \in X_{p_d}\}$
  - $Y_d = \{(p, v) | p \in OutPorts_d,\ v \in Y_{p_d}\}$
- El acoplamiento externo de entrada EIC (*external input coupling*) conecta las entradas externas con las entradas de los componentes:

$$EIC \subseteq \left\{ \big((N, ip_N), (d, ip_d)\big) | ip_N \in InPorts, d \in D, ip_d \in InPorts_d \right\}$$

- El acoplamiento externo de entrada EOC (*external output coupling*) conecta las salidas de los componentes con las salidas externas:

$$EOC \subseteq \left\{ \big((d, op_d), (N, op_N)\big) | op_N \in OutPorts, d \in D, op_d \in OutPorts_d \right\}$$

- El acoplamiento interno IC (*internal coupling*) conecta las salidas y las entradas de los componentes:

$$IC \subseteq \left\{ \big((a, ip_a), (b, ip_b)\big) | a, b \in D, ip_a \in InPorts_a, ip_b \in InPorts_b \right\}$$

  Como antes no se permite feedback directo ($a \neq b$ en la definición anterior).

- *Select* se define igual que antes.

Ejemplo 4.3. *Modelo Cola–Procesador con Puertos Consideremos el acoplamiento con puertos de la cola del Ejemplo 4.2 con un procesador como el del Ejemplo 3.2 pero también modelado con puertos.*

Un posible esquema es el de la Figura 4.4.

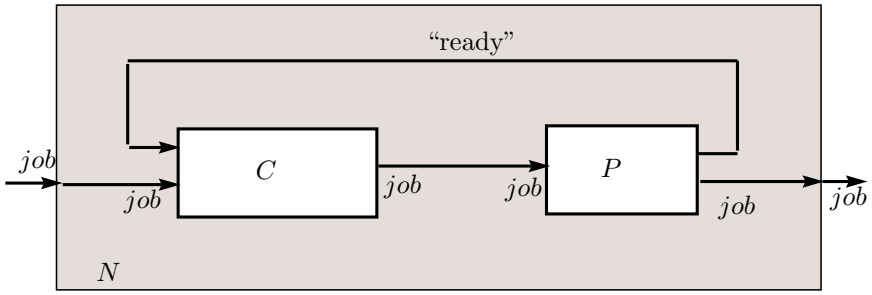

Figura 4.4: Modelo Cola–Procesador con Puertos.

La especificación del acoplamiento será:

$N = (X_N, Y_N, D, \{M_d\}, EIC, EOC, IC, Select)$, donde:
$X_N = \{inport_1\} \times J$
$Y_N = \{outport_1\} \times J$
$D = \{C, P\}$
$EIC = \left\{ \big((N, inport_1), (C, inport_1)\big) \right\}$
$EOC = \left\{ \big((P, outport_1), (N, outport_1)\big) \right\}$
$IC = \left\{ \big((P, outport_2), (C, inport_2)\big) \right\}$
$Select(\{P, C\}) = C$

4.4 Problemas Propuestos

[P4.1] Modelo Generador–Cola–Procesador.

Obtener un modelo DEVS de un sistema que produzca la siguiente secuencia de eventos:

$$\begin{aligned} t = 0 &: y = job_1 \\ t = 1 &: y = job_2 \\ t = 3 &: y = job_3 \\ t = 6 &: y = job_4 \\ t = 8 &: y = job_1 \\ t = 9 &: y = job_2 \\ &\vdots \end{aligned}$$

Luego, obtener la estructura del acoplamiento de dicho modelo con la cola y el procesador del Ejemplo 4.1 suponiendo que el generador envía los trabajos a la cola.

[P4.2] Procesamiento en Paralelo

Modificar la estructura del problema anterior agregando una nueva cola y procesador en paralelo y acoplando de forma tal que el primer grupo procese los trabajos job_1 y job_3 y que el segundo procese los trabajos job_2 y job_4.

Ayuda: Tener en cuenta que cuando una función de traducción da como resultado $Z(x) = \phi$ el evento con valor x no es transmitido (toma el valor *no evento*).

[P4.3] Procesamiento en Paralelo II

Repetir el problema anterior aprovechando el Ejemplo 4.1 y la idea del acoplamiento jerárquico.

[P4.4] Legitimidad Revisitada

Considerar dos modelos idénticos al del Ejemplo 3.1. Formar con estos un modelo acoplado conectando las salidas a las entradas (considerar que las funciones de traducción son funciones identidad).

Estudiar la legitimidad del modelo acoplado y concluir sobre la clausura bajo acoplamiento de dicha propiedad.

Ayuda: Considerar una situación inicial tal que el primer sistema está en el estado $s = 1; e = 0$ y que el segundo sistema está en $s = \infty, e = 0$.

[P4.5] Procesador con Puertos

En base al modelo del Procesador del Ejemplo 3.2 obtener un modelo con puertos como se muestra en la Figura 4.4.

Nota: Observar que aparece una dificultad ligada a la necesidad de enviar simultáneamente las señales de “ready” y la salida del trabajo.

[P4.6] Procesamiento en Paralelo III

Replantear el Problema [P4.2] acoplando mediante puertos modificando para eso el modelo del generador (y la estructura de acoplamiento por supuesto).

[P4.7] Procesamiento en Paralelo IV

Repetir el problema anterior aprovechando nuevamente la idea del acoplamiento jerárquico.

[P4.8] Integrador II

Modificar el Problema [P3.6] agregando puertos y acoplando el modelo del *integrador* a un generador que produzca la siguiente salida:

$$\begin{aligned} t = 0 &: y = 1 \\ t = 1 &: y = 2 \\ t = 3 &: y = 3 \\ t = 4 &: y = 4 \\ &\vdots \end{aligned}$$

Capítulo 5

Simulación de Modelos DEVS

Una de las características más salientes de DEVS es que modelos muy complejos pueden simularse de una manera muy simple y eficiente.

En los últimos años se han desarrollado muchas herramientas de software dedicadas a la simulación de modelos DEVS. Algunas de esas herramientas cuentan con librerías, interfaces gráficas y muchas otras facilidades para el usuario. Muchas son también gratuitas, y entre las más populares se encuentran DEVS-Java [9] y DEVSim++ [4].

Cabe mencionar aquí también una herramienta desarrollada en nuestra Facultad y que, además de ser un entorno de propósito general de simulación DEVS, tiene varias características salientes que la hacen especialmente aptas para la simulación de sistemas híbridos y para la implementación en tiempo real. Esta herramienta se llama PowerDEVS [6] y fue desarrollada como Proyecto Final de Ingeniería en la Facultad de Ciencias Exactas, Ingeniería y Agrimensura de la UNR.

Más allá de estas herramientas, los modelos DEVS pueden también simularse mediante un programa ad–hoc muy sencillo escrito en cualquier lenguaje. De hecho, la simulación de un modelo DEVS no es mucho más complicada que la simulación de un modelo de Tiempo Discreto. El único problema es que hay modelos que están compuestos por muchos subsistemas y la programación ad–hoc puede transformarse en una tarea muy tediosa.

5.1 Estructura de una Simulación de DEVS

La idea básica para la simulación de un modelo DEVS acoplado puede describirse por los siguientes pasos:

1. Buscar el modelo atómico d^* que, de acuerdo a su tiempo de avance y tiempo transcurrido, sea el próximo en realizar una transición interna.

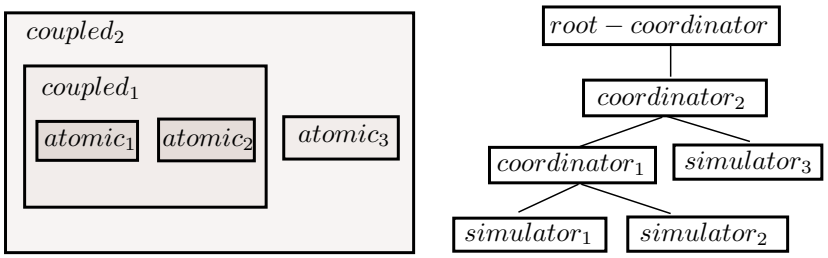

Figura 5.1: Modelo jerárquico y esquema de simulación

2. Sea tn el tiempo de la transición mencionada. Avanzar entonces el tiempo de la simulación t hasta $t = tn$ y ejecutar la función de transición interna de d^*

3. Propagar el evento de salida producido por d^* hacia todos los modelos atómicos conectados a él ejecutando las transiciones externas correspondientes. Luego, volver al paso 1

Una de las formas más simples de implementar estos pasos es escribiendo un programa con una estructura jerárquica equivalente a la estructura jerárquica del modelo a simular. De hecho, este es el método desarrollado en [8] donde una rutina llamada *DEVS–simulator* se asocia a cada modelo DEVS atómico y una otra rutina llamada *DEVS–coordinator* se relaciona a cada modelo DEVS acoplado. En la cima de la estructura jerárquica se coloca una rutina, llamada *DEVS-root-coordinator* que se encarga de avanzar el tiempo global de la simulación. La Figura 5.1 ilustra esta idea sobre un modelo DEVS acoplado.

Los simuladores y coordinadores de capas consecutivas se comunican a través de mensajes. Los coordinadores envían mensajes a sus *hijos* para que ejecuten las funciones de transición. Cuando un simulador ejecuta una transición, calcula su próximo estado y –cuando la transición es interna– envía el valor de salida a su coordinador *padre*. En todos los casos, el estado del simulador coincidirá con el estado de su modelo DEVS atómico asociado.

Cuando un coordinador ejecuta una transición, envía mensajes a algunos de sus hijos para que ejecuten sus funciones de transición correspondientes. Cuando un evento de salida producido por uno de sus hijos debe ser propagado fuera del modelo acoplado, el coordinador envía un mensaje a su propio coordinador padre con el valor de salida correspondiente.

Cada simulador o coordinador tiene una variable local tn que indica el tiempo en el que ocurrirá su próxima transición interna. En los simuladores, esa variable se calcula utilizando la función de avance de tiempo del modelo atómico correspondiente. En los coordinadores, la misma se calcula como el mínimo tn de sus hijos. Luego, el tn del coordinador que está por encima de todos es el tiempo en el cual ocurrirá el próximo evento considerando el sistema completo.

Así, el *root–coordinator* sólo debe tener en cuenta este tiempo, avanzar el tiempo global t hasta este valor y luego enviar un mensaje a su hijo para que realice la siguiente transición (y luego el *root–coordinator* repite este ciclo hasta el fin de la simulación).

5.2 Pseudo–Códigos para la Simulación de DEVS

Los siguientes pseudo–códigos describen las tres clases mencionadas en el punto anterior.

5.2.1 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
    $s = \delta_{\text{int}}(s)$
    $tl = t$
    $tn = t + ta(s)$
  when receive x-message (x, t) at time t
    $e = t - tl$
    $s = \delta_{\text{ext}}(s, e, x)$
    $tl = t$
    $tn = t + ta(s)$
end DEVS-simulator

5.2.2 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∈D}(d.tn)]
    tl = t
    tn = t + d*.tn
  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∈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
```

5.2.3 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
    s = δ_int(s)
    t = d.tn
  until end of simulation
end DEVS-root-coordinator
```

5.3 Simulación Plana

Además de la estructura de simulación detallada en las Secciones 5.1 y 5.2 hay muchas otras posibilidades para implementar la simulación de modelos DEVS.

Por ejemplo, en PowerDEVS el esquema de mensajes está invertido en lo que

refiere al tiempo de próximo evento. Aquí no es el *padre* quien mira el tiempo de próximo evento de sus *hijos* sino estos últimos quienes lo informan. De esta forma, es posible detectar y tratar interrupciones en tiempo real a nivel de los modelos atómicos.

Más allá de estas variantes, el principal problema con la metodología descripta es que, debido a la estructura jerárquica, puede haber un importante tráfico de mensajes entre las capas más altas y las más bajas.

Puede verse por ejemplo que en el modelo de la Figura 5.1, la transmisión de un evento desde el modelo $atomic_1$ hacia el modelo $atomic_3$ implica mensajes del $simulator_1$ al $coordinator_1$, de este último al $coordinator_2$, y recién entonces al $simulator_3$.

Todos estos mensajes y su tiempo computacional correspondiente pueden evitarse con el uso de una estructura de simulación plana, en la cual todos los simuladores están a un mismo nivel jerárquico bajo un único coordinador.

La forma de transformar una simulación jerárquica en una plana es muy simple en el contexto de DEVS y agrega mucha eficiencia [3]. De hecho, la mayor parte de las herramientas de software mencionadas implementan la simulación en base a un código plano.

5.4 Problemas Propuestos

Bibliografía

[1] François Baccelli, Guy Cohen, Geert Olsder, y Jean Pierre Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[2] Christos Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. Irwin and Aksen, 1993.

[3] Kihyung Kim, Wonseok Kang, y Hyungon Seo. Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. En *Proceedings of Anual Simulation Symposium*, 2000.

[4] Tag Gon Kim. *DEVSim++ User's Manual. C++ Based Simulation with Hierarchical Modular DEVS Models*. Korea Advance Institute of Science and Technology, 1994. Disponible en http://www.acims.arizona.edu/.

[5] Ernesto Kofman. *Simulación y Control de Sistemas Continuos por Eventos Discretos.* Tesis Doctoral, Facultad de Ciencias Exactas,Ingeniería y Agrimensura., 2003.

[6] Esteban Pagliero y Marcelo Lapadula. Herramienta Integrada de Modelado y Simulación de Sistemas de Eventos Discretos. Proyecto Final de Ingeniería. FCEIA, UNR, Argentina, Julio 2002.

[7] B. Zeigler y S. Vahie. DEVS Formalism and Methodology: Unity of Conception/Diversity of Application. En *Proceedings of the Winter Simulation Conference*, Washington D.C., 1993.

[8] Bernard Zeigler, Tag Gon Kim, y Herbert Praehofer. *Theory of Modeling and Simulation. Second edition*. Academic Press, New York, 2000.

[9] Bernard Zeigler y Hessam Sarjoughian. *Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations*. Arizona Center for Integrative Modeling and Simulation. Disponible en http://www.acims.arizona.edu/.

[10] Bernard Zeigler. *Theory of Modeling and Simulation*. John Wiley & Sons, New York, 1976.