AADECA 2008 - Semana del Control Automático - XXIº Congreso Argentino de Control Automático
1 al 3 de Septiembre de 2008 - Buenos Aires, Argentina.

DESARROLLO DE UN SIMULADOR DE SISTEMAS HÍBRIDOS EN TIEMPO REAL

Federico Bergero [1] **Ernesto Kofman** [2]
Cristian Basabilbaso Juan Zúccolo

Laboratorio de Sistemas Dinámicos. FCEIA - UNR.
CIFASIS–CONICET. Riobamba 245 bis - (2000) Rosario

Resumen: En este trabajo presentaremos el desarrollo de una nueva herramienta de simulación en tiempo real. Utilizando como base la herramienta PowerDEVS de simulación de sistemas de eventos discretos, adaptando su código para el funcionamiento en un sistema operativo de tiempo real (Linux RTAI) e incorporando al motor de simulación nuevas funcionalidades orientadas al cumplimiento de restricciones temporales y de interconexión con el mundo exterior, obtuvimos una plataforma de simulación que es capaz de cumplir con diferentes requisitos asociados a la simulación en tiempo real. Presentamos también diferentes estudios de performance de la nueva plataforma y algunos resultados de aplicación.

Palabras Claves: DEVS, Tiempo Real, Simulación, Linux

1. INTRODUCCIÓN

La simulación digital se ha tornado en una herramienta casi imprescindible para el diseño, análisis, optimización, control, detección de fallas y en muchos otros problemas ligados a la ingeniería, la física, la química, etc.

Dependiendo de la naturaleza de los procesos a simular, hay múltiples formalismos descriptivos de los modelos. A grandes rasgos, siguiendo la definición de Zeigler *et al.* (2000), estos formalismos se pueden clasificar en tres categorías: sistemas continuos (ecuaciones diferenciales, ordinarias y parciales), sistemas de tiempo discreto (ecuaciones en diferencias) y sistemas de eventos discretos (redes de Petri, autómatas, DEVS, etc.).

Dentro de los formalismos nombrados, DEVS (Zeigler *et al.*, 2000) cuenta con algunas particularidades. En principio, este formalismo permite representar cualquier sistema que realice un número finito de cambios en intervalos finitos de tiempo. Por esto, mediante DEVS, se puede representar cualquier tipo de sistemas de eventos discreto y de tiempo discreto. Más aún, teniendo en cuenta que la simulación de sistemas continuos en general requiere de algún tipo de discretización, es posible simular mediante DEVS sistemas continuos aproximados mediante cualquier método de integración numérica.

En otras palabras, DEVS es la herramienta más general de simulación de sistemas discretos, continuos e híbridos en general. Esta generalidad ha impulsado el desarrollo de múltiples herramientas de simulación basadas en DEVS. Entre las más difundidas, podemos mencionar a CD++ (Wainer, 2002), DEVS-Java (Zeigler and Sarjoughian, 2000), JDEVS (Filippi and Bisgambiglia, 2004) y PowerDEVS (Pagliero *et al.*, 2003).

Entre estas herramientas, PowerDEVS tiene la particularidad de estar fuertemente orientada a la simulación de sistemas continuos e híbridos en general, al implementar de manera completa la familia de los métodos de QSS (Cellier and Kofman, 2006), consistentes en métodos de aproximación de ecuaciones diferenciales mediante DEVS. Teniendo una interface gráfica de edición de diagramas de bloques, PowerDEVS se asemeja mucho (desde el punto de vista del usuario) a Simulink, aunque su motor de simulación se basa en principios totalmente distintos.

Muchas aplicaciones requieren que las simulaciones se realicen en *tiempo real*, es decir, que el tiempo de simulación se mantenga lo más próximo posible al tiempo físico (Cellier and Kofman, 2006). Ejemplos de estas aplicaciones incluyen a los sistemas *hardware in the loop* (cuando la simulación debe interactuar con algún dispositivo de hardware, para detección de fallas, monitoreo, prueba de controladores, observadores, etc.) y los sistemas *man in the loop* (cuando la simulación

[1] bergero@cifasis-conicet.gov.ar
[2] kofman@fceia.unr.edu.ar

interactúa con una persona, como por ejemplo en los simuladores de vuelo).

Las restricciones impuestas por este tipo de simulación abren una serie de problemas de relativa complejidad que involucran cuestiones de sistema operativo, entrada–salida de datos, interrupciones, etc. Por esto, el desarrollo de una herramienta de simulación de propósito general de tiempo real es una tarea compleja, pero que a su vez tiene una aplicabilidad muy importante y amplia.

En este trabajo, presentamos entonces el desarrollo de una nueva plataforma de simulación en tiempo real, consistente esencialmente en la adaptación de PowerDEVS a un sistema operativo de tiempo real –Linux RTAI (Mantegazza *et al.*, 2000)– y a la incorporación de todas las funcionalidades que se requieren de un simulador de tiempo real. De esta manera, teniendo en cuenta las simulitudes entre PowerDEVS y Simulink, la herramienta resultante tiene cierta semejanza con el Real Time Workshop de Simulink (aunque nuevamente basado en principios de simulación diferentes). El desarrollo realizado fue parte de una Tesina de Licenciatura en Ciencias de la Computación de la UNR (Bergero, 2008).

El trabajo está organizado como sigue: en la Sección 2 describimos los principios del formalismo DEVS y algunos conceptos de simulación en tiempo real. Luego, en la Sección 3 presentamos el grueso del trabajo realizado y por último, en la Sección 4 mostramos algunas aplicaciones y resultados experimentales obtenidos usando esta nueva herramienta.

2. MODELOS DEVS, SIMULACIÓN Y TIEMPO REAL

Presentaremos en esta sección algunos conceptos teóricos, sobre el lenguaje formal DEVS y sobre el simulado de modelos DEVS, tanto formalmente como llevado a la práctica en tiempo real.

2.1 Formalismo DEVS

DEVS es un formalismo para modelar y analizar sitemas de eventos discretos(es decir, sistemas en los cuales en un lapso finito de tiempo, ocurren una cantidad finita de cambios).

Un modelo DEVS puede ser visto como un autómata que procesa una serie de eventos de entrada y genera una serie de eventos de salida. Este procesamiento está regido por la estructura interna de cada uno de las partes que componen el modelo general.

Formalmente un modelo DEVS está descripto por dos clases de componentes, modelos atómicos y modelos acoplados.

Un modelo atómico representa la unidad "molecular" de procesamiento, en el sentido que es la pieza fundamental y más básica. Formalmente un modelo atómico esta conformado por la 7-upla

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

Cada posible estado s ($s \in S$) tiene asociado un *Avance de Tiempo* calculado por la *Función de Avance de Tiempo* $ta(s)$. 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 $s_2 = \delta_{int}(s_1)$. La función δ_{int} se llama *Función de Transición Interna*.

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 ($\lambda : S \rightarrow Y$) se llama *Función de Salida*. 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 $s_4 = \delta_{ext}(s_3, e, x_1)$ (notar que $ta(s_3) > e$). En este caso se dice que el sistema realiza una *transición externa*. La función δ_{ext} se llama *Función de Transición Externa*. Durante una transición externa no se produce ningún evento de salida.

La descripción de un sistema puede ser (formalmente hablando) completamente realizada utilizando modelos atómicos, aunque esto resulta un poco incómodo y confuso. Para abordar este problema, el formalismo DEVS introduce lo que se llaman *modelos acoplados* que es una forma de agrupar modelos DEVS y generar nuevos modelos a partir de este agrupamiento.

Los modelos DEVS pueden acoplarse de forma equivalente a los diagramas de bloques, a través de puertos de entrada y salida. Los modelos acoplados son en sí mismo modelos DEVS válidos, formalmente el acoplamiento es una operación cerrada sobre el conjunto de modelos DEVS. Acoplar modelos DEVS forma nuevos modelos DEVS.

Por lo tanto, los modelos DEVS pueden acoplarse de manera jerárquica, es decir, los modelos acoplados pueden a su vez acoplarse con otros modelos acoplados y/o atómicos.

2.2 Simuladores de Modelos DEVS

La simulación de modelos DEVS puede ser llevada a cabo fácilmente en cualquier computadora. Para simular el funcionamiento de un sistema DEVS sólo debemos realizar los siguientes pasos:

- Buscar el modelo (pertenciente al modelo acoplado raíz), que de acuerdo a su función de avance ta y el tiempo transcurrido, es el próximo a realizar una transición interna. A este modelo, lo llamaremos d^* y t_n al tiempo de su próxima transición interna.
- Avazamos el tiempo de simulación t a t_n y "ejecutamos" la transición interna del modelo d^*.
- Luego, propagamos el evento de salida producido por la transición interna recién realizada por d^* a los modelos conectados a d^*, ejecutando la transición externa de cada uno de estos.
- Finalmente volvemos a realizar el ciclo desde el primer paso

2.3 PowerDEVS

PowerDEVS (Pagliero *et al.*, 2003) es un entorno de simulación de modelos DEVS de propósito general. Fue desarrollado originalmente en la Facultad de Ciencias Exactas, Ingeniería y Agrimensura de la Universidad Nacional de Rosario como proyecto final.

La implementación de esta herramienta está dividida en dos módulos bien heterogéneos.

Entorno de desarrollo gráfico: Es la interfaz que ve el usuario regular de PowerDEVS. Permite describir modelos DEVS de una forma gráfica y sencilla. Está implementado en Visual Basic y corre sobre Windows. Este módulo es el encargado de generar el código C++ que luego será compilado y ejecutado para simular el modelo DEVS correspondiente. La interfaz gráfica consiste esencialmente en un editor de diagramas de bloques con una funcionalidad muy similar a la de Simulink.

Motor de simulación: Este módulo es el núcleo de toda la herramienta. El código que es generado por el entorno gráfico, luego es compilado junto con el motor de simulación y se obtiene el programa que simula el modelo DEVS. Su implementación está hecha en C++ por lo cual puede ser portado fácilmente a cualquier arquitectura y sistema operativo.

Una de las características salientes de PowerDEVS es que contiene una librería completa de implementación de los métodos de QSS (Cellier and Kofman, 2006), los cuales realizan una aproximación DEVS de sistemas continuos. De esta manera, PowerDEVS permite simular sistemas continuos y discretos (es decir, sistemas híbridos) de una manera muy simple y eficiente.

2.4 Sistemas Operativos de Tiempo Real

Los sistemas operativos de tiempo real son sistemas operativos que ofrecen las herramientas básicas para poder implementar tareas con restricciones temporales. Un sistema opreativo debe ofrecer métodos de expresar las restricciones temporales de cada tarea, métodos de comunicación entre estas tareas y manejo de los recursos de bajo nivel de la computadora (memoria, interrupciones, puertos, etc). Entre los más populares encontramos QNX (Hildebrand, 1992), VxWorks (Wehner, 2006), Drops (Härtig *et al.*, 1998), RT-Linux (Barabanov, 1997), RTAI (Mantegazza *et al.*, 2000).

Entre estos sistemas, RT-Linux y RTAI nos resultaron los más apropiados, ya que son una extensión del kernel de Linux, es decir todo el software desarrollado para GNU/Linux corre sobre este sistema. Esto representa una ventaja ya que existen muchas aplicaciones, documentación, y una gran comunidad de usuarios. La empresa que desarrolla RT-Linux (FMSLabs) libera una versión gratis (RT-Free) y otra paga. Pero impone restricciones de uso sobre la versión libre.

Por esta razón nuestra elección fue utilizar RTAI, ya que el desarrollo no debería tener restricciones de uso ni de distribución.

2.5 Caracterísiticas de Linux RTAI

RTAI (RealTime Aplication Interface) (Mantegazza *et al.*, 2000) es un sistema operativo de tiempo real que soporta varias arquitecturas, entre ellas i386 (PC). Más especificamente es una extensión del sistema operativo Linux (Bovet and Cesati, 2002)(el cual no es de tiempo real). Esta extensión del kernel de Linux brinda al usuario las herramientas necesarias para desarrollar sistemas en tiempo real, entre ellas destacamos:

- Respuesta determinística a interrupciones.
- Comunicación entre Procesos.
- Timers de alta precisión.
- Manejo de interrupciones.

Como dijimos antes RTAI es una extensión del kernel de Linux, por lo cual todo el "bagage" de software desarrollado para Linux (entornos gráficos, herramientas matemáticas) puede ser conectado a la simulación. También al tener licencia GPL tenemos la libertad de distribuirlo sin restricciones.

3. POWERDEVS EN TIEMPO REAL

El motor de simulación de PowerDEVS es una implementación en C++ del simulador planteado en (Zeigler *et al.*, 2000). Al estar desarrollado en un lenguaje portable, no hubo que hacer cambios significativos en el código generado por la interfaz gráfica, sino que las modificaciones fueron introducidas en las clases fundamentales del simulador.

Estas modificaciones fueron realizadas mediante el agregado de cuatro módulos que brindan al usuario los servicios de sincronización, captura y tratamiento de interrupciones, manejo de archivos y medición del tiempo real.

Además, se desarrolló una librería con modelos atómicos que utilizan estos servicios de manera que un usuario neófito pueda usar las herramientas sin necesidad de programar nada.

3.1 Módulo de Sincronización

Este módulo es la piedra fundamental del desarrollo. Ofrece al motor el servicio de sincronización con el reloj de RTAI y posee la lógica de cómo implementar esta sincronización.

El sistema operativo RTAI nos provee varios métodos de sincronización con precisión de ηs. Entre estos métodos tenemos dos que son los que utilizamos *rt_sleep* y *rt_busy_sleep*. El primero implementa una espera en la cual el procesador es liberado y dado a otro proceso (de tiempo real o no). El segundo implementa un busy-waiting, es decir consume ciclos de CPU esperando que el tiempo transcurra. Se obtiene una sincronización más precisa con este método a costa de monopolizar el uso del procesador.

En base a estos métodos, el módulo ofrece un servicio de sincronización al usuario de PowerDEVS con dos modalidades: *Normal* y *Precisa*. Este servicio es utilizado para que el tiempo en el que se emiten los eventos del modelo se corresponda con el tiempo real. El usuario accede a este servicio declarando que un determinado evento a emitir es de tiempo real, pudiendo elegir en ese caso la modalidad.

El servicio de sincronización lo que hace es esperar hasta que el tiempo real alcanza al tiempo de simulación y está implementado de la siguiente forma:

- Si la espera es demasiado chica (menor que cierta constante), la espera no se realiza[3].
- Si la espera pedida es del tipo *Normal*, se realiza una espera *no-busy*, es decir, se libera el procesador.
- Si la espera es una espera *Precisa*, primero vemos la magnitud de la espera pedida.
  - Si esta magnitud es menor que cierta constante, realizamos una espera *busy* y luego devolvemos el control a la simulación.
  - Si esta magnitud es mayor, realizamos la mayor parte de la espera como una espera *no-busy* y luego compensamos el faltante con una espera *busy*. De esta forma, al realizar un espera precisa liberamos el procesador y obtenemos buena precisión.

[3] Este límite está impuesto para no realizar esperas que en la mayoría de los casos, conllevarían a un error mayor que no esperar

3.2 Módulo de Interrupciones

Al integrar conceptos de tiempo real al simulador de PowerDEVS, se nos abre un campo extensamente amplio en la simulación y control de sistemas mediante modelos DEVS. Para aprovecharlo, se debe integrar PowerDEVS con conceptos de Hardware de la PC como puertos, interrupciones, direcciones de memoria, etc. El incoveniente es que lo que se simula en DEVS es un modelo cerrado, no hay agentes externos (como se requiere para este caso), es decir en todo momento se sabe cual va a ser la próxima transición. Esto se contradice al escenario donde uno puede estar esperando que un usuario aprete una tecla o que un dispositivo externo envíe datos asincrónicamente. La acción de un evento externo al modelo DEVS no puede ser representado en el mismo lenguaje del formalismo DEVS, debe entonces tomar riendas el motor de simulación para captar este evento.

Desarrollamos para este fin, el módulo de interrupciones, el cual se encarga de capturar las interrupciones cuando ocurren y avisar a los modelos que les interesen la interrupción en cuestión.

Un atómico puede expresar (generalmente en su método de inicialización) que está interesado en un interrupción mediante el método:

```
requestIRQ(int irq);
```

Cuando ocurra la interrupción *irq*, el motor de simulación emulará la llegada de un evento al atómico que pidió ser informado de esta interrupción. El atómico se verá obligado a ejecutar su método de transición externa. Los avisos de interrupción llegan desde un puerto ficticio con numeración -1 (un puerto no válido para los atómicos desarrollados en PowerDEVS).

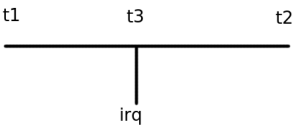

Fig. 1. Tiempo de interrupción

Previo a este aviso, el tiempo de simulación debe ser re-ajustado. Si el próximo evento a emitir debe ocurrir en t_2 (Fig.1) pero antes de alcanzar ese tiempo ocurre una interrupción en el momento t_3 el tiempo de simulación debe ser llevado hacia atrás hasta t_3. El evento que iba a ser emitido en t_2 es descartado.

Para la implementación de este módulo utilizamos servicios de RTAI que dan la posibilidad de esperar una interrupción de hardware. Esta interrupción es capturada antes por RTAI que por Linux. Luego, un proceso instalado como *handler* de esa interrupción (si es que el modelo está interesado en esa interrupción) es ejecutado, y notifica al atómico en cuestión.

3.3 Módulo de Archivos

Cuando uno simula algo, espera analizar los resultados con algún fin. Evidentemente la simulación debe dejar o mostrar los resultados por algún medio al usuario. En la versión de Windows de PowerDEVS esto se logra a través de salida escritas a un archivo o por algún medio gráfico (de plotting).

Esto es una necesidad básica para el motor de simulación por lo cual la versión de tiempo real de PowerDEVS debe incluir algun método como los de la versión de Windows. Aunque parezca simple, el problema surge debido a que los procesos de tiempo real no pueden utilizar llamdas al kernel de Linux, privándolo así del uso del *file system*, entorno gráficos, etc. RTAI ofrece métodos de comunicación entre procesos (IPC). En base a estos métodos de IPC, implementamos una interfaz al *file system* de Linux. Funciona de la siguiente manera

- Al inicar la simulación se crea un proceso en Linux (es decir no en tiempo real)
- Se crea un *pipe* desde el proceso de tiempo real al proceso en Linux (la dirección indica quien escribe y quien lee en el pipe).
- Cuando el proceso de tiempo real (el que ejecuta la simulación) requiere escribir algo al *file system*, lo escribe en el pipe y su trabajo está realizado. En RTAI esto se realiza sin retardos ni demoras (como si sucedería bajo Linux).
- Cuando el proceso de Linux es ejecutado (cuando no hay nada que hacer en tiempo real), lee del *pipe* (si es que hay datos) y los escribe al archivo del filesystem de Linux. Una vez terminado este ciclo intenta leer nuevamente del *pipe* y queda a la espera de que el proceso en tiempo real vuelva a escribir datos.
- Cuando la simulación es terminada el proceso en tiempo real "avisa" al proceso de Linux que no escribirá más datos y ambos procesos terminan.

3.4 Módulo de medición del tiempo

Agregamos también un pequeño módulo que permite a los modelos atómicos conocer el tiempo real para poder realizar acciones distintas, por ejemplo, dependiendo de cuán desfasado está el tiempo de simulación con el tiempo real. Aunque en principio, el tiempo real debería estar siempre sincronizado con el tiempo de simulación, no es así el caso ante la ocurrencia de una interrupción o frente a problemas de overrun (es decir, cuando los cálculos llevan más tiempo que el tiempo real disponible para realizarlos).

3.5 Librería de Tiempo Real

Para utilizar todas las ventajas de tiempo real de una forma más simple y sencilla agregamos a la libería de PowerDEVS varios bloques elementales para incluir directamente en los modelos.

RTWait: Este atómico tiene como objetivo sincronizar todos los eventos que pasan por él. Así un modelo desarrollado a la vieja usanza (sin sincronización) puede ser fácilmente convertido en un modelo que sincronize los eventos en tiempo real. El atómico debe ser puesto en la conexión sobre la cual se quiere sincronizar los eventos. El atómico tiene un parámetro para elegir qué clase de sincronización se desea realizar, *precisa* o *normal* (ver 3.1). Este atómico es simplemente un filtro, en el cual los eventos de entrada se emiten (instantáneamente) pero como eventos sincronizados.

RTClock: Este modelo es un modelo tipo filtro. Cuando recibe un evento externo, emite instantáneamente un evento que tiene como valor el tiempo real. Utiliza los servicios que ofrece el módulo de medición de tiempo real.

GNUPlot: Este modelo permite reproducir de forma gráfica los resultados de la simulacion.

ToDisk: Este modelo escribe los eventos recibidos a un archivo en disco en formato CSV(Comma Separated Values).

IRQDetector: Este modelo es la interface de usuario con el módulo de interrupciones de PowerDEVS. Emite un evento cuando ha ocurrido la interrupción correspondiente.

4. EJEMPLOS Y RESULTADOS

Para validar la plataforma, realizamos algunos ejemplos prácticos, probando así todas las nuevas funcionalidades (sincronización temporal, escritura a puertos, detección de interrupciones, etc).

Primero mostraremos algunas las mediciones realizadas sobre el simulador y luego algunos ejemplos de aplicación. Los ejemplos fueron ejecutados sobre una PC AMD Athlon de 1.8 GHz.

4.1 Mediciones de Latencia de Sincronización

Una de las características importantes a la hora de implementar un sistema de tiempo real, es saber cuál es el *"error"* (latencia) de sincronización de los eventos, es decir, la diferencia entre el tiempo de reloj y el tiempo de emisión.

En este caso, hicimos el siguiente experimento: Utilizamos primero como modelo una fuente que genera la aproximación QSS de una onda senoidal de 440Hz (este modelo genera 20 eventos por ciclo, esto es, 8800 eventos por segundo). Simulamos el modelo en tiempo real y medimos la latencia máxima y la promedio (en general lo más importante es la máxima, que representa el peor caso). Obtuvimos los siguientes resultados:

Tabla 1. Error de sincronización sin carga

-	Máximo	Promedio	Overruns
Normal	4800 ηs	1500 ηs	-
Precisa	450 ηs	180 ηs	-

Luego realizamos mediciones con una carga computacional mucho mayor, utilizando como ejemplo la simulación de un control proporcional de un motor de corriente continua mediante PWM (*Pulse Width Modulation*). En este caso, fuimos incrementando la frecuencia de la portadora triangular del PWM desde los 1000 hasta los 20000 Hz. Obtuvimos los siguientes resultados:

Tabla 2. Error de sincronización con carga (N.=Normal, P.=Preciso)

f (Hz)	Esp.	Máx.	Prom.	Overruns
1000	N.	5786 ηs	1512 ηs	-
	P.	1330 ηs	180 ηs	-
15000	N.	5622 ηs	1622 ηs	372 / 19905 ev.
	P.	1000 ηs	512 ηs	305 / 19905 ev.
17000	N.	4547 ηs	1648 ηs	6119 / 17616 ev.
	P.	973 ηs	454 ηs	5924 / 17616 ev.
20000	N.	0 ηs	0 ηs	18292 / 18292 ev.
	P.	0 ηs	0 ηs	18292 / 18292 ev.

Como vemos en la tabla 2, con la frecuencia de 1000 Hz no hay *overruns*, es decir, no hay cálculos que superen el tiempo que tenían disponibles para realizarse.

Entre los 15000 Hz y 17000 Hz, además de los *overruns* se observan retrasos en el sistema Linux (la parte que no es tiempo real) tanto en la respuesta interactiva como en el refresco de pantalla, lo que indica que nos estamos acercando al límite de carga del sistema de tiempo real.

Aumentando el valor de la frecuencia a 20000 Hz llegamos a sobrecargar el sistema de tiempo real a un punto tal que no se realiza ninguna espera, ya que todos los eventos sufren *overruns*, o sea que el valor a emitir se termina de calcular después del momento en el cual debía ser emitido.

Este último caso refleja una limitación de la plataforma de Hardware más que del sistema de sincronización. Lo que ocurre es que no se alcanzan a realizar los cálculos en el tiempo disponible. Los únicos errores atribuibles al sistema de tiempo real son los de latencia.

4.2 Medición de Latencia de Interrupción

Para medir la latencia de interrupción, utilizamos un procedimiento bastante simple. Creamos un modelo (Fig.2) que genera interrupciones y luego las captura, midiendo el lapso de tiempo entre que la interrupción ocurre y que el controlador (handler) se ejecuta para atenderla.

Para ello utilizamos el puerto paralelo de la computadora. Uno de los pines del puerto paralelo (STO) es el encargado de generar la interrupción. Para medir la latencia entonces procedemos como sigue:

- Contectamos el pin STO a un pin de datos previamente llevado a 0 (escribiendo 0 en el puerto paralelo).
- "Instalamos" en el handler correspondiente al puerto paralelo, una función, *onIrq*.
- Tomamos el tiempo actual (con la mayor precisión que tengamos) y lo "guardamos" en una variable.
- Escribimos 1 en el puerto de datos del puerto paralelo, generando así una interrupción.
- Al generarse la interrupción el sistema ejecutará la función *onIrq*, tomamos el tiempo actual y le restamos el tiempo previo que habíamos guardado.

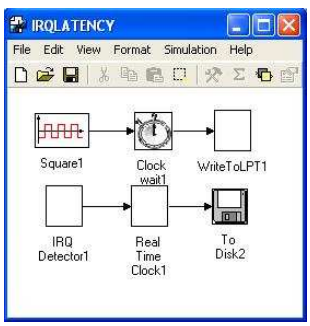

Fig. 2. Modelo de detección de interrupciones

Utilizando esta método obtuvimos latencias de interrupción de 20μs.

4.3 Audio por Puerto Paralelo

La primera aplicación que realizamos fue la de emitir sonido a través del puerto paralelo. Lo que nos propusimos es producir una onda de sonido modulada mediante PWM, utilizando así un sólo bit de salida del puerto. La onda la formamos sumando tres senoidales de distinta frecuencia, lo que formaría un acorde (ver Fig.3). El modelo de PowerDEVS se muestra en la Fig.4. Al simular el modelo, capturamos además el sonido con un micrófono, obteniendo una forma de onda bastante similar a la original como puede verse en la Fig.5.

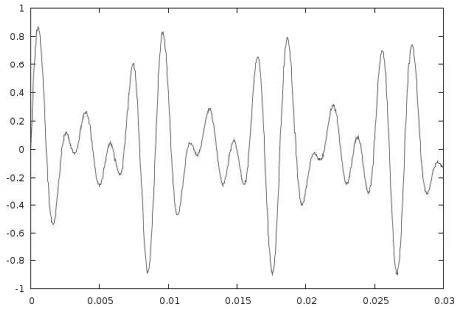

Fig. 3. Onda a emitir

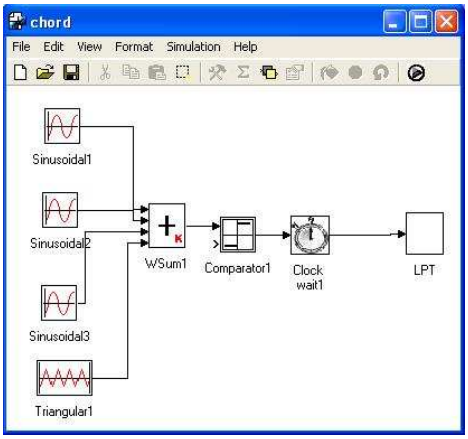

Fig. 4. Modelo a simular

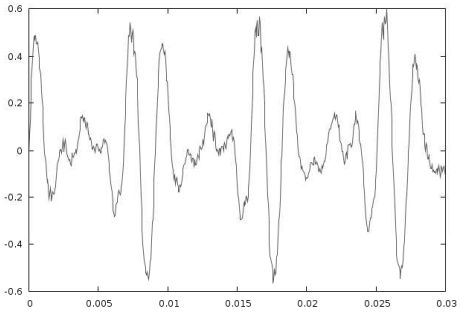

Fig. 5. Onda capturada por un micrófono

4.4 Control de Motor de Corriente Continua

Otra aplicación que implementamos fue la del control de un motor de corriente continua. Para llevar a cabo este experimento, diseñamos un dispositivo de hardware (ver Fig.6), compuesto por un motor y un encoder (implementado utilizando un mouse de PC). Los pulsos del encoder se introducen en el bit *STO* del puerto paralelo, lo que provoca interrupciones en la PC. Por otro lado, el actuador, consistente en un amplificador implementado con un transistor y un filtro de salida, toma la tensión de un pin de datos del puerto paralelo.

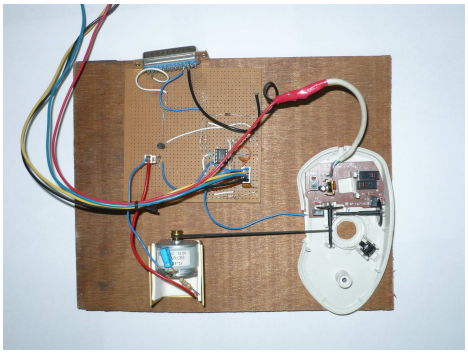

Fig. 6. Motor y mouse

El control se realiza de la siguiente forma (ver Fig.7). El bloque *Motor Speed* detecta y cuenta las interrupciones para estimar la velocidad. Esta medición se compara con la referencia (calculada por el bloque *WSum4*) y se realiza un control proporcional integral. La señal de control (salida del PI) se modula mediante PWM (previo saturación para evitar sobremodulación). La señal de PWM (dado por el bloque *Comparator1*), se envía al puerto paralelo una vez sincronizada.

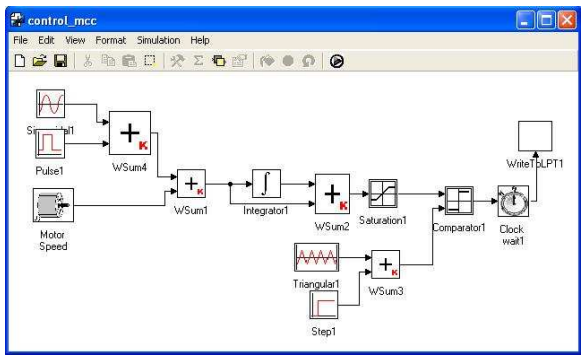

Fig. 7. Modelo del control proporcional

La Fig.8 grafica la señal de referencia mientras que la Fig.9 grafica la velocidad medida.

5. CONCLUSIONES

En el presente trabajo desarrollamos una nueva herramienta para la simulación de sistemas híbridos en tiempo real basado en el formalismo DEVS.

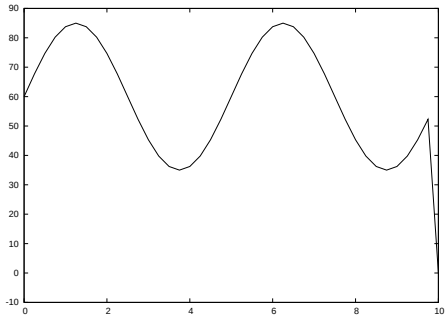

Fig. 8. Velocidad de referencia

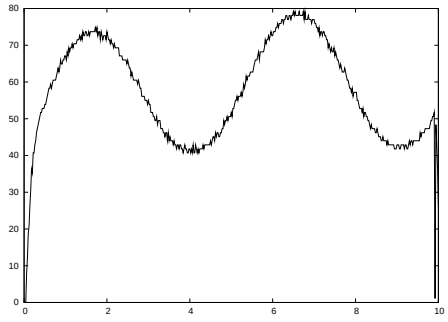

Fig. 9. Velocidad medida

Esta herramienta puede ser utilizada en muchísimas aplicaciones principalmente de ingeniería (procesamiento de señales, implementación de control, de observadores, detección de fallas, etc.).

La herramienta garantiza una latencia máxima del orden de 2 μs (4 órdenes de magnitud de mejora frente a lo que se puede obtener utilizando Windows). De esta manera, podríamos emitir eventos a una frecuencia de 500 KHz (suponiendo que el Hardware tiene velocidad suficiente para realizar los cáculos involucrados)

Mediante varios ejemplos de aplicación, verificamos el correcto funcionamiento de la herramienta y su aplicabilidad en casos reales (particularmente en el caso del control del motor de corriente continua) .

Actualmente estamos trabajando en utilizar esta herramienta como base de distintas aplicaciones prácticas, particularmente para el procesamiento asincrónico de señales en tiempo real y para el desarrollo de una plataforma de prueba para sistemas de control de movimiento (y de electrónica de potencia en general).

Ponemos especial énfasis en las aplicaciones de control de electrónica de potencia ya que los métodos de QSS que utiliza PowerDEVS tienen una gran eficiencia para la simulación en tiempo real de esta clase de sistemas.

En este momento estamos también finalizando una versión distribuible de la herramienta con el sistema operativo de tiempo real completo. Actualmente, la herramienta (que como ya mencionamos es totalmente gratuita) puede descargar del sitio: www.fceia.unr.edu.ar/lsd/powerdevs.

REFERENCIAS

Barabanov, Michael (1997). A linux-based real-time operating system. Master's thesis. New Mexico Institute of Mining and Technology. New Mexico.

Bergero, Federico (2008). Desarrollo de una plataforma de simulación en tiempo real por eventos discretos. Tesina de Grado. FCEIA – UNR.

Bovet, Daniel and Marco Cesati (2002). *Understanding the Linux Kernel*. O' Reilly.

Cellier, Francois E. and Ernesto Kofman (2006). *Continuous System Simulation*. Springer. New York.

Filippi, Jean-Baptiste and Paul Bisgambiglia (2004). Jdevs: an implementation of a devs based formal framework for environmental modelling. *Environmental Modelling & Software* **19**(3), 261–274.

Härtig, Hermann, Robert Baumgartl, Martin Borriss, Claude-Joachim Hamann, Micheal Hohmuth, Frank Mehnert, Lars Reuther, Sebastian Schönberg and Jean Wolter (1998). Drops: Os support for distributed multimedia applications. In: *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*. pp. 203–209.

Hildebrand, Dan (1992). An architectural overview of qnx. In: *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*. Berkeley, CA, USA. pp. 113–126.

Mantegazza, P., E. L. Dozio and S. Papacharalambous (2000). Rtai: Real time application interface. *Linux Journal*.

Pagliero, Esteban, Marcelo Lapadula and Ernesto Kofman (2003). PowerDEVS. Una Herramienta Integrada de Simulación por Eventos Discretos. In: *Proceedings de RPIC 2003*. Vol. 1. San Nicolas, Argentina. pp. 316–321.

Wainer, Gabriel (2002). Cd++: a toolkit to develop devs models. *Software: Practice and Experience* **32**(13), 1261–1306.

Wehner, Christof (2006). *Tornado and VxWorks*. BoD.

Zeigler, Bernard and Hessam Sarjoughian (2000). *Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations*. Arizona Center for Integrative Modeling and Simulation. Available at http://www.acims.arizona.edu/.

Zeigler, Bernard, Tag Gon Kim and Herbert Praehofer (2000). *Theory of Modeling and Simulation. Second edition*. Academic Press. New York.