POWERDEVS. UNA HERRAMIENTA INTEGRADA DE SIMULACIÓN POR EVENTOS DISCRETOS

Esteban PAGLIERO†, Marcelo LAPADULA† y Ernesto KOFMAN† ††

†*Departamento de Electrónica, FCEIA - UNR.* ††*CONICET*
Riobamba 245 bis - (2000) Rosario - Argentina
Email: apaono@siderar.com – m.lapadula@ieee.org – kofman@fceia.unr.edu.ar

***Resumen*— En este artículo se presenta una nueva herramienta de software para la simulación de sistemas dinámicos basada en el formalismo DEVS (Discrete Event System Specification). La herramienta, denominada *PowerDEVS*, permite la creación de modelos DEVS elementales en código C, la edición gráfica del acoplamiento de los mismos y la posterior simulación, brindando también la posibilidad de crear librerías de modelos. Además de la simplicidad de su utilización, una de las características salientes de la herramienta es la posibilidad de realizar simulaciones en tiempo real. De esta forma, además de permitir simular sistemas, permite implementar controladores digitales tanto sincrónicos como asincrónicos.**

***Palabras Clave*— Simulación, DEVS, Sistemas de Eventos Discretos, Software.**

I. INTRODUCCIÓN

DEVS (Zeigler, 1976; Zeigler *et al.*, 2000) es el formalismo más general para la descripción de sistemas discretos. Mediante el mismo es posible representar cualquier sistema que realice un número finito de cambios en un intervalo finito de tiempo. De esta forma, no sólo las Redes de Petri, Statecharts, Grafcets y Event–Graphs sino también todos los sistema de tiempo discreto pueden verse como casos particulares de DEVS.

Teniendo en cuenta que los sistemas continuos pueden aproximarse mediante sistemas de tiempo discreto –a través de métodos numéricos– y que éstos son un caso particular de DEVS, es claro que DEVS puede también aproximar sistemas continuos. Más aún, hay métodos numéricos –como QSS y QSS2 (Kofman y Junco, 2001; Kofman, 2002a)– que producen modelos de simulación que no pueden representarse en tiempo discreto sino únicamente como modelos DEVS.

Por esto, las herramientas de simulación de modelos DEVS son potencialmente mucho más generales que las de los distintos formalismos discretos e incluso que las de tiempo continuo y tiempo discreto como Simulink (Matlab), Scicos (Scilab), etc.

Entre las herramientas de simulación de DEVS se destacan DEVS–Java (Zeigler y Sarjoughian, 2000), DEVSim++ (Kim, 1994), DEVS–C++ (Cho y Cho, 1997), CD++ (Wainer *et al.*, 2001) y JDEVS (Filippi *et al.*, 2002).

Los simuladores basados en Java (DEVS–Java y JDEVS) ofrecen buenas interfaces gráficas para la descripción del acoplamiento, pero son inferiores en cuanto a eficiencia computacional cuando son utilizados en una PC en comparación a los restantes, basados en C++.

En cuanto a estos últimos, el principal problema es que no tienen interfaces gráficas y la descripción del acoplamiento se torna muy trabajosa.

Además, todos los programas nombrados son herramientas desarrolladas desde y hacia el ambiente de las Ciencias de la Computación, bajo el concepto de que el usuario es un programador avanzado.

En virtud de esto, *PowerDEVS* fue concebido como una herramienta computacionalmente eficiente y flexible (basada en C++), capaz de brindar una interfaz apropiada tanto para un programador experto como para una persona sin conocimientos de programación. Por otro lado, dado que fue desarrollada desde un ambiente de Ingeniería, se orientó su funcionalidad de manera tal que resulte similar a entornos populares como Simulink, brindando facilidades para la parametrización de modelos previamente disponibles y para la creación de nuevos modelos.

Aparte de estas ventajas, se invirtió el esquema clásico de simulación de DEVS (Zeigler *et al.*, 2000) a modo de permitir la simulación en tiempo real con captura de interrupciones provenientes del entorno real para poder así implementar controladores de todo tipo. De esta forma, *PowerDEVS* se convierte también en la primera herramienta capaz de implementar directamente esquemas de control asincrónico QSC (Kofman, 2003).

El trabajo se organiza de la siguiente forma: Tras introducir el formalismo DEVS en la Sección I, en la Sección II se presentan las principales características del entorno *PowerDEVS*, tanto en su parte funcional como en algunos aspectos de su implementación. Finalmente, la Sección III presenta algunos ejemplos y resultados de simulación con la herramienta seguidos de las conclusiones correspondientes.

II. FORMALISMO DEVS

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.

A. Modelos DEVS atómicos

Formalmente, 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* $ta(s)$ ($ta(s) : S \to \mathbb{R}_0^+$). 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*.

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. 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 entonces *Función de Transición Externa*. Durante una transición externa no se produce ningún evento de salida.

B. Modelos DEVS acoplados

Como ya fue mencionado, DEVS es un formalismo muy general y puede describir sistemas muy complejos. Sin embargo, la representación de un sistema complejo basado solamente en funciones de transición y de avance de tiempo es demasiado difícil. La razón es que todas las situaciones posibles en el sistema deben ser pensadas y descriptas por dichas funciones.

Afortunadamente, los sistemas complejos pueden en general pensarse como el acoplamiento de sistemas más simples. Mediante el acoplamiento, los eventos de salida de algunos subsistemas se convierten en eventos de entrada de otros subsistemas. La teoría garantiza que el acoplamiento de modelos DEVS define un nuevo modelo DEVS (o sea, DEVS es cerrado frente al acoplamiento) y los sistemas complejos pueden ser representados por DEVS de una manera jerárquica.

Cuando se utiliza acoplamiento mediante puertos de entrada y salida, los modelos DEVS acoplados son simples Diagramas de Bloques donde cada bloque puede ser un modelo DEVS atómico o bien un nuevo acoplamiento de bloques.

C. 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.
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 (Zeigler *et al.*, 2000) 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.

III. EL ENTORNO POWERDEVS

PowerDEVS es un conjunto de herramientas de software que corren en el entorno Windows. Estas permiten editar modelos DEVS especificando el acoplamiento de manera gráfica y las funciones de los modelos atómicos en lenguaje C++.

Al dar la orden de simular, se generan una serie de archivos de código fuente que se compilan y enlazan produciendo un programa ejecutable que lleva al cabo la simulación correspondiente.

Este conjunto de herramientas puede separarse en dos módulos bien definidos, el entorno de edición de modelos y el generador de código de simulación.

A. Edición de Modelos

Las herramientas de edición son el editor de modelos propiamente dicho y el editor de atómicos.

El editor de modelos funciona de manera muy similar a Simulink, permitiendo editar y parametrizar Diagramas de Bloques, invocar la simulación y construir librerías.

Dicho editor consta de una ventana principal (Fig.1) cuya tarea fundamental es administrar las librerías de bloques además de permitir el acceso a las opciones de configuración del programa y dar las funciones básicas de crea y abrir modelos.

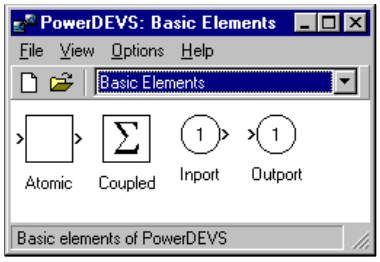

Figura 1: Ventana principal mostrando una librería.

Las ventanas de edición de modelos en tanto (Fig.2) brindan el espacio para construir gráficamente los modelos, permitiendo la completa parametrización de los mismos para su simulación directa.

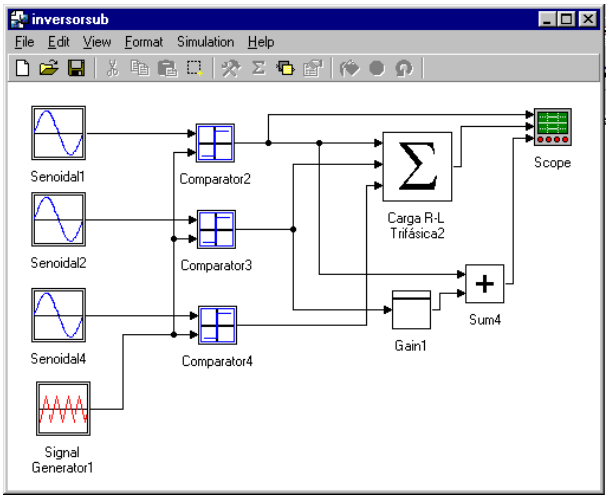

Figura 2: Ventana de edición de modelos.

Las librerías se visualizan en la ventana principal o en las ventanas de librería y se pueden crear o editar como modelos (exportándolas como librerías).

Los modelos se construyen arrastrando y soltando los elementos y puertos desde la ventana principal o las ventanas de librería hacia las ventanas de edición. También pueden copiarse y pegarse un conjunto de objetos seleccionados del mismo modelo o de otros modelos. La característica multidocumento hace que se pueda trabajar con varios modelos a la vez.

Las líneas de conexión se crean y modifican también de manera gráfica, clickeando y arrastrando con el mouse.

Utilizando la opción correspondiente del menú *Edit* –o su versión emergente con el botón derecho del mouse– se pueden editar las propiedades de los bloques mediante la ventana de edición de estos elementos (Fig.3).

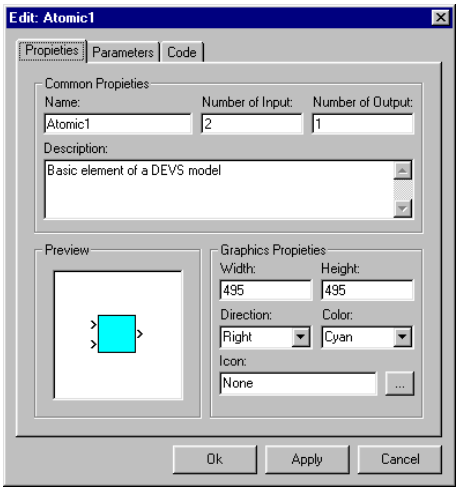

Figura 3: Ventana de edición de bloques.

Dicha ventana de edición de los bloques posee tres solapas:

Properties: Permite editar las principales propiedades gráficas de los bloques.

Parameters: Sobre esta se indica que parámetros (y de que tipo) caracterizan al bloque. Estos parámetros –que pasan a formar parte de las funciones de transición– pueden luego modificarse directamente al hacer doble click sobre el bloque.

Inputs/Outputs: Esta pertenece únicamente a los acoplados y en esta se determina el orden en que aparecen los puertos de entrada y salida transformados en entradas y salidas del acoplado, respectivamente.

Code: Esta pertenece únicamente a los atómicos y determina la ruta del archivo de cabecera que posee las definiciones y el código C++ que modela el comportamiento del atómico, permitiendo también editar dicho código invocando al editor de atómicos (Fig.4).

El editor de atómicos es una aplicación independiente del entorno de edición de modelos y automatiza las tareas de generación de los archivos de código fuente y archivos de cabecera a partir de la definición de las funciones de inicialización, de transición interna y externa, de salida y de avance de tiempo.

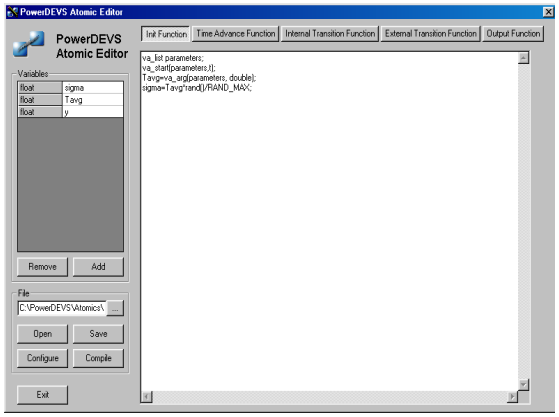

Figura 4: Ventana del editor de atómicos.

B. Simulación

PowerDEVS es un software que automatiza la programación de modelos DEVS en C++ de una forma totalmente transparente al usuario. Siempre y cuando el usuario no pretenda definir nuevos modelos atómicos, no se requiere ningún tipo de conocimiento de dicho lenguaje.

La herramienta trata cada modelo atómico como una clase derivada de una clase abstracta *simulator*, la cual resuelve las cuestiones relativas a la simulación. Las clases que representan a un atómico implementan las funciones del formalismo DEVS (transición interna, transición externa, salida y avance de tiempo) y una función de inicialización para establecer condiciones iniciales y capturar los parámetros.

Como se mencionó antes, el código de cada función lo introduce el usuario en el editor de atómicos y es este mismo programa el que crea las clases automáticamente a partir de dicho código.

La coordinación entre las clases elementales de acuerdo a la estructura del modelo la resuelve el Pre–Procesador cuya finalidad es convertir en C++ la definición estructural del modelo dada por el Diagrama de Bloques del Editor de Modelos.

El Pre-Procesador utiliza esta definición de la estructura para construir un objeto que representa al modelo. Esta clase que representa al modelo es la que implementa computacionalmente la simulación.

Además de la implementación computacional del modelo, el Preprocesador genera una interfaz de usuario que permite acceder a los distintos modos de simulación.

Todo este código es entonces compilado con lo que se genera un archivo ejecutable que presenta la interfaz mencionada y realiza la simulación.

En la Fig.5 se observa la interfaz de usuario del Pre-Procesador. Esta cuenta con las acciones de generar el código y de compilar y ejecutar.

En el ángulo superior izquierdo de la Fig.6 se observa la interfaz de simulación, que permite seleccionar los

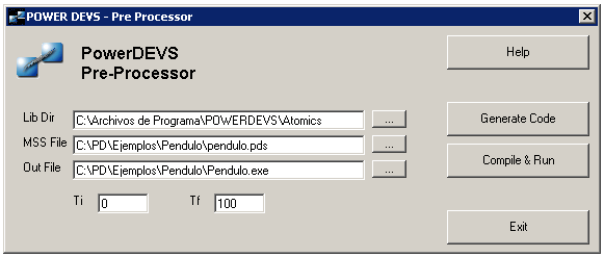

Figura 5: *PowerDEVS* Pre-Processor

tres modos de ejecución:

Run: Simulación en forma continua del modelo.

Tiemed: Simulación temporizada (*Tiempo Real*)

Step: Simulación paso a paso.

En la Fig.6 también se observa la ventana de progreso de simulación y un graficador.

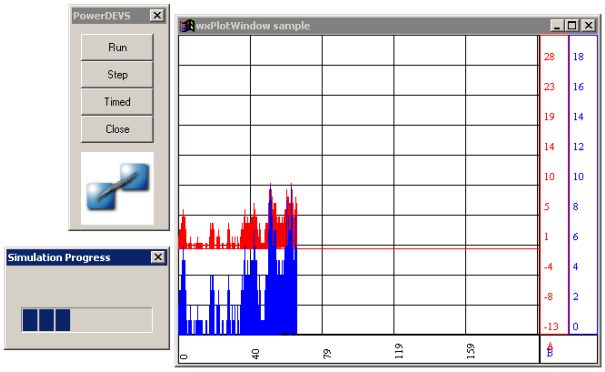

Figura 6: *PowerDEVS* ejecutando una simulación

Si bien el editor de modelos sólo funciona bajo Windows, el código generado por el Preprocesador es independiente de la plataforma incluso el de los componentes gráficos. Para estos se utilizaron las librerías *wxWindows*, disponibles para una gran cantidad de plataformas como ser Windows, Linux, Solaris, MAC, etc.

Con la misma filosofía se adoptó como compilador el GNU. Ambos componentes son de libre distribución como *PowerDevs* lo que permite su completa utilización con costo nulo.

El uso de las librerías gráficas antes mencionadas permite además la inclusión de atómicos, ya sea de entrada o de salida, que contengan elementos gráficos. Esto brinda la posibilidad de la creación de modelos atómicos que contengan interfaces de usuario fáciles y prácticas para su uso. Uno de estos ejemplos es el graficador (a la derecha en la Fig.6) que se incluye con las librerías estándar. También puede programarse muy fácil un conjunto de botones desde los cuales el usuario opere un sistema para control en tiempo real.

C. Simulación en Tiempo Real

PowerDevs tiene la capacidad de ejecutar simulaciones temporizadas y además fue diseñado para permitir el procesamiento de eventos externos. Esto, sumado a la flexibilidad que nos provee un lenguaje como C++, permite la simulación en tiempo real.

Esto se simplifica además debido a que desde los modelos atómicos es posible acceder a librerías externas mediante las cuales se puede interactuar con diferentes dispositivos del sistema donde se ejecuta la simulación (puertos, interrupciones, etc.).

La performance de las simulaciones en tiempo real en diferentes pruebas ha sido satisfactoria si bien la precisión de la temporización así como la velocidad dependen en gran medida de la plataforma donde se ejecute.

IV. EJEMPLOS Y RESULTADOS

A. Inversor Trifásico

En este ejemplo se modelizó y simuló un inversor senoidal trifásico que utiliza el método de suboscilación como estrategia de control alimentando a una carga R-L simétrica.

Este método también es conocido como *PWM* senoidal. La señal de referencia es una senoidal que se compara con una portadora triangular generandose una señal lógica que comanda las columnas del convertidor de potencia.

Para la simulación se construyó el modelo mostrado en la Fig.2, seleccionándose una frecuencia de portadora $f_s = 750Hz$ y una frecuencia fundamental igual a la frecuencia de línea $f_l = 50Hz$.

En el modelo de la carga (Fig.7) puede observarse la parte continua del sistema, donde los boques integradores y las ganancias utilizan el método QSS2 de integración de ecuaciones diferenciales.

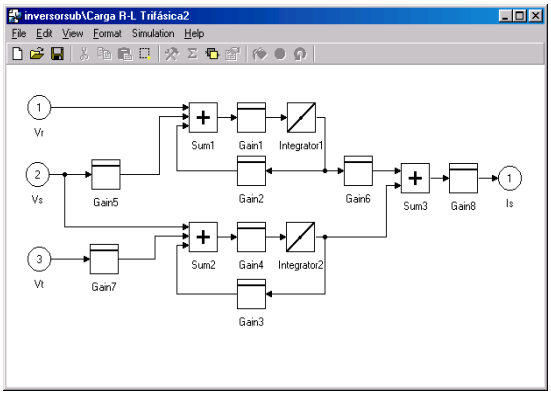

Figura 7: Modelo de la carga.

La Fig.8 muestra una parte de la trayectoria de la corriente de carga obtenida tras la simulación, tal cual como se visualiza en el bloque *scope* de la Fig.2. Dicho bloque es también un modelo atómico DEVS que incluye rutinas gráficas de visualización.

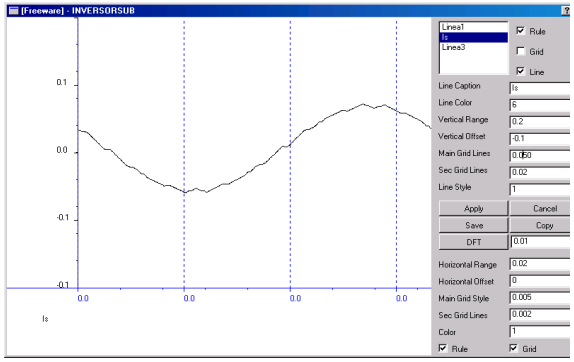

Figura 8: Corriente de carga.

B. Control de un Péndulo Invertido.

El siguiente ejemplo simula el sistema de control digital QSC de un péndulo invertido propuesto en (Kofman, 2002b). La planta consiste entonces en el modelo linealizado de un péndulo invertido y el controlador continuo original fue diseñado mediante LQR.

La implementación digital QSC de dicho controlador consiste en la aproximación del mismo por el método de QSS. Esto es equivalente a reemplazar los integradores del controlador por integradores cuantificados. Además, el esquema QSC implica también el uso de conversores A/D y D/A asincrónicos.

De esta forma, el modelo de simulación es el mostrado en la Fig.9, donde los subsistemas indicados como *Péndulo* y *Controlador* contienen los Diagramas de Bloques correspondientes.

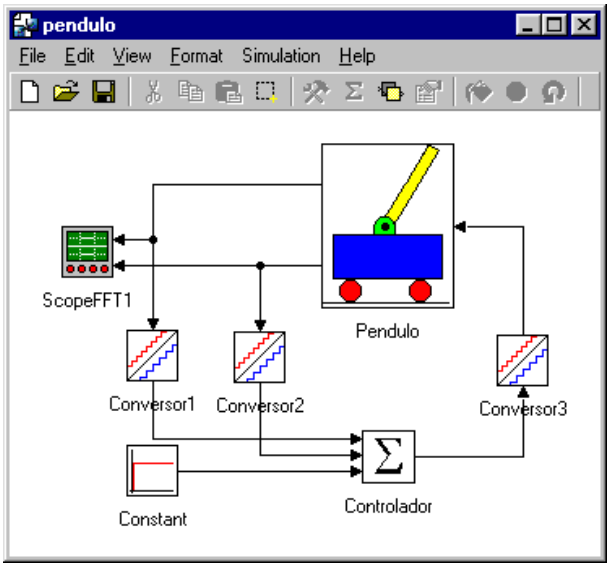

Figura 9: Sistema de control QSC en *PowerDEVS*.

El modelo del Controlador y de los conversores es exacto (ya que DEVS simula exactamente los QSS). La parte continua en cambio (la planta) debe aproximarse mediante algún método numérico. En este caso se utilizó también QSS ya que dicho método tiene grandes

ventajas cuando se utiliza con sistemas híbridos como este. Además, se utilizó dicho método con un quantum muy chico para que el error introducido por el mismo sea despreciable frente a los efectos de cuantificación de los conversores y del controlador QSC.

Se simuló entonces la respuesta a un escalón de referencia de 0.2m en la posición del carro a partir del reposo. Los resultados de simulación se salvaron en un archivo y se importaron con Matlab. Esto permitió graficarlos y compararlos con los resultados obtenidos mediante el controlador continuo y un controlador digital clásico.

Los resultados se muestran en las Fig.10.

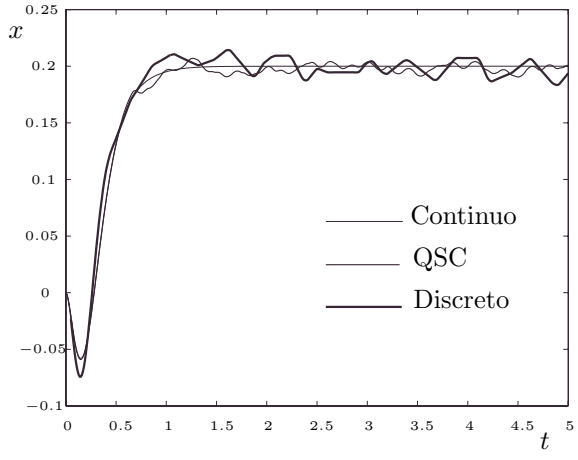

Figura 10: Posición del carro.

Es importante remarcar que este ejemplo no puede simularse con Matlab/Simulink. Si bien el controlador QSC puede describirse muy fácilmente en términos de DEVS, su representación y simulación es prácticamente imposible sin este formalismo.

V. CONCLUSIONES

Se presentaron aquí las principales características y aplicaciones de *PowerDEVS*. De las mismas se desprende que la herramienta es capaz de brindar soluciones muy simples y eficientes en una gama muy amplia de problemas de simulación y control.

La simplicidad y flexibilidad para la definición de nuevos bloques atómicos y librerías brindan al software posibilidades casi ilimitadas para aplicaciones en muchísimas ramas de la técnica.

Además de esto, las ventajas respecto a otras herramientas similares existentes incluyen la amigabilidad del entorno y la capacidad de realizar simulaciones en tiempo real (incluyendo captura de interrupciones).

Con respecto a trabajo futuro, se está intentando migrar el código del editor de modelos de Visual Basic a C++. La idea de esto es que todo el programa sea de código abierto y pueda correr bajo cualquier plataforma.

Otra herramienta que se quiere incorporar es una etapa de *aplanamiento* de código para mejorar la eficiencia de las simulaciones, especialmente en tiempo real. Esto se puede lograr fácilmente aplanando directamente la estructura jerárquica (Kim *et al.*, 2000).

Dejando a un lado estas mejoras, la programación de nuevos modelos atómicos y la creación de nuevas librerías de modelos correspondientes a diferentes áreas es lo que en definitiva va a potenciar el uso y el campo de aplicación de *PowerDEVS*. Por esto, la principal tarea a futuro es la de generar estas nuevas librerías.

REFERENCIAS

Cho, H. y Y. Cho (1997). *DEVS–C++ Reference Guide*. The University of Arizona.

Filippi, J., M. Delhom, y F. Bernardi, "The JDEVS Environmental Modeling and Simulation Environment," *Proceedings of IEMSS 2002* **3**, 283–288 (2002).

Kim, K., W. Kang, S. B., y H. Seo, "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non–Hierarchical One," *Proc. of the 33rd Annual Simulation Symposium* 227–233 (2000).

Kim, T. G. (1994). *DEVSim++ User's Manual. C++ Based Simulation with Hierarchical Modular DEVS Models*. Korea Advance Institute of Science and Technology.

Kofman, E., "A Second Order Approximation for DEVS Simulation of Continuous Systems," *Simulation* **78**(2), 76–89 (2002a).

Kofman, E., "Quantized-State Control of Linear Systems," *Proceedings of AADECA 2002*, Buenos Aires, Argentina (2002b).

Kofman, E., "Quantized-State Control. A Method for Discrete Event Control of Continuous Systems.," *Latin American Applied Research* **33**(4), 399–406 (2003).

Kofman, E. y S. Junco, "Quantized State Systems. A DEVS Approach for Continuous System Simulation," *Transactions of SCS* **18**(3), 123–132 (2001).

Wainer, G., G. Christen, y A. Dobniewski, "Defining DEVS Models with the CD++ Toolkit," *Proceedings of ESS2001* 633–637 (2001).

Zeigler, B., *Theory of Modeling and Simulation*, John Wiley & Sons, New York (1976).

Zeigler, B., T. Kim, y H. Praehofer, *Theory of Modeling and Simulation. Second edition*, Academic Press, New York (2000).

Zeigler, B. y H. 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.