Un Modelo de Eventos Discretos para la Simulación de Sistemas de Tiempo Real

Francisco E. Paez[1], Jose M. Urriza[1], Javier D. Orozco[2], Carlos E. Buckle[1]

[1] Universidad Nacional de la Patagonia San Juan Bosco
Puerto Madryn, Argentina
[2] Universidad Nacional del Sur
Bahia Blanca, Argentina
{franpaez, josemurriza, jadorozco, carlos.buckle}@gmail.com

Resumen. En este trabajo se presenta un modelo de eventos discretos para el diseño de simuladores de Sistemas de Tiempo Real. Los simuladores sirven para realizar comprobaciones de algoritmos, modelos, técnicas y para evaluar métricas de rendimiento. El modelo de eventos discretos se ajusta perfectamente a los sistemas dinámicos discretos como lo son los Sistemas de Tiempo Real. El modelo se desarrolla empleando la técnica de grafo de eventos, debido a su sencillez y facilidad de implementación.

Palabras clave: Sistemas de Tiempo Real, Simulación, Modelado, Eventos Discretos.

1. Introducción

En la actualidad el uso de la simulación por computadora es esencial en un gran número de disciplinas. La misma es un soporte fundamental para la obtención de nuevos resultados en investigaciones, acelerando los procesos de creación de nuevos métodos y técnicas. Sin embargo, pocas veces se pone énfasis en la revisión y validación del *software* empleado para llegar a cabo los resultados publicados. Por otro lado, es evidente la importancia que posee para otros grupos de investigación, el poder validar dichos resultados, mediante la reproducción de los experimentos realizados, empleando las mismas soluciones de *software* u otras que sigan los mismos lineamientos y diseños.

Este trabajo presenta un modelo de eventos discretos para la simulación de Sistemas de Tiempo Real (*STR*). El objetivo es contar con un modelo marco, sobre el cual desarrollar sistemas de simulación por computadora. Este modelo ha sido empleado con éxito en el simulador de *STR* del grupo de investigación de la UNPSJB[1].

[1] Grupo de investigación en *STR* de la Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB) Sede Puerto Madryn (http://www.rtsg.unp.edu.ar).

En el pasado, un conjunto de aplicaciones y *frameworks* han sido desarrollados para la simulación de *STR*. Se pueden mencionar, entre las más conocidas, a STRESS ([1]), PERTS ([2]), YASA ([3]), Cheddar ([4]), RealTTS ([5]) y el simulador de la Université Libre de Bruxelles ([6]). Se ha encontrado también aplicaciones como MAST ([7]) que ofrecen herramientas de modelado, o FORTISSIMO ([8]) que ofrece un *framework* para el diseño de simulaciones. Un trabajo que estudia los *STR* como *sistemas discretos* es [9], donde son modelados mediante redes de Petri, a fin de permitir análisis temporales eficientes. También en [10], donde se presenta un marco formal general para el estudio de *STR*.

El trabajo se encuentra organizado de la siguiente manera: en la sección 2 se realiza una breve introducción y análisis de los *STR*. En la sección 3, se presenta una introducción a la simulación por eventos discretos y a la técnica de modelado *grafo de eventos*. En la sección 4 se desarrolla el modelo de eventos discretos y en la sección 5 se describe una implementación de referencia. Finalmente, las conclusiones, junto con posibles trabajos futuros, son discutidas en la sección 6.

2. Introducción a los *STR*

Una definición formal de los *STR*, ampliamente aceptada en la disciplina, es la formulada por Stankovic en [11], la cual dice: "*En los STR los resultados no solo deben ser correctos aritmética y lógicamente, si no que, además, deben producirse antes de un determinado tiempo, denominado vencimiento*".

Dependiendo de cuan crítico es el cumplimiento de este vencimiento, se puede clasificar a los *STR* en tres tipos. El primer tipo no tolera la perdida de ningún vencimiento, y se los denomina *duros* o *críticos*. El segundo tipo permite la pérdida de algunos vencimientos, por lo cual se los conoce como *blandos*. Finalmente, los de tercer tipo, denominados *firmes*, tipifican las pérdidas según algún criterio estadístico.

En los *STR duros* la pérdida de un vencimiento puede tener graves consecuencias para la integridad del sistema y, posiblemente para su entorno. Este tipo de *STR* es utilizado en aviónica, sistemas de control industrial, soporte a la vida, etc. Debido a su criticidad, es necesario realizar un *análisis de planificabilidad* para garantizar, a priori, el cumplimiento de las *constricciones temporales* de todas las tareas.

El marco de trabajo para sistemas mono-recurso y multitarea fue presentado en [12], donde las tareas son consideras periódicas, independientes y apropiables. Las tareas pueden ser asignadas al recurso mediante una asignación estática o basada en prioridades. Esta última es la más utilizada, y emplea una *política de prioridades* que permite al planificador determinar que tarea utilizará el recurso en un instante dado.

Bajo este marco de trabajo, una tarea i (τ_i) de tiempo real (*TR*) es parametrizada mediante su peor tiempo de ejecución (C_i), periodo (T_i) y vencimiento (D_i). Luego, un conjunto de n tareas de *TR* es especificado como $\Gamma(n) = \{(C_1, D_1, T_1), ..., (C_n, T_n, D_n)\}$. Se define $c_i(t)$ como el tiempo ejecutado de la tarea i al instante t.

En [12] se demostró también, que el peor estado de carga para un planificador mono-recurso es aquel en que todas las tareas solicitan ejecución en un mismo instante. A este instante se lo denomina *instante crítico*. Si el planificador logra

ejecutar todas las instancias de las tareas que llegaron en el instante crítico, antes de su vencimiento, el *STR* se dice *planificable* y consecuentemente lo es para cualquier otro estado de carga.

2.1. Caracterización y Análisis de Sistemas de Tiempo Real

Un *sistema dinámico* es aquel cuyo estado evoluciona a través del tiempo. Consecuentemente, un *STR* es un sistema dinámico. Un análisis de los *STR* como *sistemas dinámicos,* cuando los mismos son planificados por *Rate Monotonic* (*RM*, [12]) o *Deadline Monotonic* (*DM*, [13]), ha sido realizado en [14]. En el mismo, se presenta la evolución del *STR* a partir del *instante crítico*, utilizando el método de cálculo del peor tiempo de respuesta de una tarea, por medio de una ecuación de *Punto Fijo* (*PF*), descrita en [15]:

$$t^{q+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t^q}{T_j} \right\rceil C_j \qquad (1)$$

Esta ecuación permite, modelar y calcular, mediante un proceso iterativo, la evolución del *subsistema* $\Gamma(i)$ desde el *instante crítico*. El método iterativo se detiene si encuentra un *PF* tal que $t^q = t^{q+1} \leq D_i$, luego el sistema es planificable. Si el *PF* se encuentra en un instante posterior al vencimiento de la tarea analizada, es *no planificable*, $t^{q+1} > D_i$. En sistemas sobresaturados puede no existir un *PF* tal que $t^q = t^{q+1}$. La ecuación (1) muestra las siguientes características:

- Es monótona creciente.
- Describe la evolución de un sistema dinámico.
- Es determinística, ya que para cada valor de t existe un único estado posible.
- La función techo ($\lceil\ \rceil$) introduce no-linealidad.

Esto permite concluir que un *STR* es un sistema *dinámico*, *no-lineal*, *discreto* y *determinístico* ([14]), cuando el mismo es diagramado mediante alguna disciplina de prioridades fijas como por ejemplo *RM* o *DM*.

3. Simulación por Eventos Discretos

A continuación se presenta una introducción al paradigma de *simulación por eventos discretos*, y a la técnica de modelado *grafo de eventos*.

La *simulación por eventos discretos* (*Discrete Event Simulation*, *DES*) es aplicada en el estudio de sistemas que pueden ser representados mediante modelos lógico-matemáticos discretos[2]. En los mismos las variables de interés cambian de manera instantánea, sólo en un determinado número contable de instantes precisos, dado un intervalo determinado de tiempo ([16]). Se define como *evento* (v), al conjunto atómico de modificaciones sobre las variables de estado del sistema en un instante determinado.

[2] El sistema original puede ser tanto *discreto* como *continuo*.

Los modelos empleados son generalmente representados mediante relaciones recursivas, como por ejemplo $t^{k+1} = 2t^k$, donde k denota los pasos discretos en el tiempo, o como la ecuación (1).

Durante la simulación se cuenta con un *reloj de simulación*, t, que registra el instante de tiempo actual, y una lista de eventos futuros Λ. Esta es un conjunto de tuplas (t_i, v_i), siendo $t_i \geq t$ el instante en donde se ejecutará el evento v_i. Generalmente Λ es implementado como una lista de prioridades, en base a los valores t_i.

En cada paso de la simulación, se ejecuta el primer evento v_i en Λ (el de mayor prioridad), y se actualiza t con el valor t_i asociado a v_i. Procesos concurrentes pueden ser simulados fácilmente mediante múltiples eventos planificados en un mismo t_i. La simulación comienza con un conjunto inicial de eventos, generalmente con $t_i = 0$. Finaliza cuando t supera un cierto valor determinado (t_{end}), se encuentra un evento especial de finalización (v_{end}) o Λ no contiene más elementos ($\Lambda = \varnothing$).

3.1. Grafos de Eventos

Un modelo de eventos discretos puede diseñarse mediante la técnica de *grafo de eventos* ([17, 18]). La dinámica del sistema es caracterizada mediante *eventos*, que representan las modificaciones al estado del sistema. Las relaciones lógicas y temporales entre los mismos son indicadas mediante arcos. Es importante resaltar que el grafo de eventos no representa un autómata.

Un modelo de *grafo de eventos* $\boldsymbol{M}$ consiste de los siguientes componentes:

- $\boldsymbol{S}$, un conjunto de variables que definen el estado del sistema.
- $\boldsymbol{V}$, un conjunto de vértices que representan los eventos de interés en el sistema.
- $\boldsymbol{E}$, un conjunto de arcos dirigidos $e_{od} = (v_o, v_d)$ (*evento origen-destino*) que indican las relaciones temporales y lógicas entre pares de eventos en $\boldsymbol{V}$.
- $\boldsymbol{F} = \{f_v : \boldsymbol{S} \rightarrow \boldsymbol{S} \; \forall \; v \in \boldsymbol{V}\}$, funciones asociadas a cada vértice $v \in \boldsymbol{V}$, que describen los cambios en $\boldsymbol{S}$, producto de la ocurrencia del evento.
- $\boldsymbol{C} = \{c_{od} : \boldsymbol{S} \rightarrow \{0,1\} \forall e_{od} \in \boldsymbol{E}\}$, las condiciones asociadas a cada arco e_{od}. La transición e_{od} es válida si y sólo si c_{od} es evaluada con éxito ($c_{od} = 1$).
- $\boldsymbol{D} = \{\delta_{od} \in \mathbb{R}_0^+ \forall e_{od} \in \boldsymbol{E}\}$ los *deltas* (tiempos de retraso) asociadas a cada arco e_{od}.
- $\boldsymbol{A} = \{\boldsymbol{A_e}, e_{od} \in E\}$, atributos asociados al arco e_{od}.
- $\boldsymbol{B} = \{\boldsymbol{B_v}, v \in V\}$, parámetros asociados al evento v.

Luego un modelo de grafo esta especificado como $\boldsymbol{M} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{F}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{B})$. Cada arco dirigido $e_{od} = (v_o, v_d)$ se recorre si y sólo si la condición c_{od}, es válida luego de ejecutar el evento v_o. Recorrer el arco equivale a programar la ejecución del evento v_d en el instante $t + \delta_{od}$, siendo δ_{od} el *delta* asociado al arco e_{od}.

Dado un arco e_{od}, el conjunto de atributos $\boldsymbol{A_e}$ asociado serán los argumentos formales que el evento v_d recibirá como parámetros (conjunto $\boldsymbol{B_v}$). En caso de no requerirse parámetros $\boldsymbol{A}$ y $\boldsymbol{B}$ son conjuntos vacíos.

Es importante notar que tanto Λ como t están asociados con la ejecución de la simulación de $\boldsymbol{M}$, y no son partes propiamente dichas del modelo. Esta técnica será empleada a continuación para realizar el modelado de un *STR* como un *sistema de eventos discretos*.

4. Modelado de un *STR* mediante eventos discretos

A continuación se desarrolla el modelo de eventos discretos de un *STR*, mediante la técnica de *grafo de eventos*. El modelo identifica los arribos de nuevas instancias y la programación de los eventos de ejecución, desalojo y finalización.

4.1. Eventos

El modelo del *STR* que se propone cuenta con seis eventos. El primer evento, v_0, corresponde con el *Inicio* del *STR* (tiempo de *setup*). Para una instancia cualquiera de una tarea de *TR*, se identifican los eventos *Arribo* (v_1), *Ejecución* (v_2), *Finalización* (v_3) y *Desalojo* (v_4). El evento v_1 recibe como parámetro la tarea del *STR* a ejecutar. Finalmente, el evento *Terminación* (v_5) indica el fin de la ejecución de la simulación. Es planificado en el instante t_{end} en donde se desea que la misma finalice.

4.2. Prioridades de los eventos

Durante la ejecución de la simulación es posible que dos o más eventos se encuentren planificados en un mismo instante *t*. Es necesario realizar una asignación de prioridades sobre los eventos. De esta manera, el orden de ejecución de los mismos será coherente para los fines de la simulación. Las prioridades son indicadas en la Tabla 1. El máximo nivel de prioridad es 0.

Tabla 1. Prioridades de Ejecución de los Eventos

Evento	Prioridad
Inicio (v_0)	0
Terminación (v_5)	1
Finalización (v_3)	2
Desalojo (v_4)	3
Arribo (v_1)	4
Ejecución (v_2)	5

4.3. Arcos

Los eventos v_0 a v_4 son conectados por 6 arcos (Figura 1):

- Arco e_{01}: programa los arribos de la primera instancia de cada tarea del *STR*.
- Arco e_{11}: planifica el próximo evento v_1 (*Arribo*).
- Arco e_{12}: programa el evento v_2 (*Ejecución*).
- Arco e_{23}: realiza la planificación del evento v_3 (*finalización*).
- Arco e_{24}: programa el evento v_4 (*Desalojo*).
- Arco e_{32}: planifica un nuevo evento v_2 (*Ejecución*).

4.4. Condiciones

El arco e_{01} se recorre una única vez, al planificar los eventos v_1 iniciales, al inicio de la simulación. Por lo tanto no cuenta con una condición asociada. A continuación se presenta cada condición c_{od} asociada al arco e_{od}, que debe de cumplirse para la programación del evento destino (v_d):

- Condición c_{11}: La instancia anterior de la tarea τ_i no excedió su tiempo de ejecución, por lo tanto se programa el próximo arribo de la tarea.
- Condición c_{12}: No existen eventos v_1 en Λ (arribos de nuevas instancias de tareas de *TR*) en el instante actual, por lo tanto se pasa a programar el evento v_2.
- Condición c_{23}: La instancia actual puede finalizar su ejecución antes del arribo del próximo evento v_1 más próximo.
- Condición c_{24}: La instancia actual no finaliza su ejecución antes del arribo del evento v_1 más próximo, por lo tanto es desalojada del recurso.
- Condición c_{32}: No existen eventos v_1 en Λ para el instante actual y hay al menos una tarea de *TR* que no ha finalizado su ejecución en la cola de listos del planificador.

La condición c_{32} evita la duplicación de un evento v_2, en caso de existir un evento v_1 en el mismo instante. Las condiciones c_{23} y c_{24} son mutuamente excluyentes.

4.5. Tiempos de retraso

La planificación de un nuevo evento v_i debe realizarse para un instante $t_i \geq t$. Para el modelo de *grafo de eventos*, esto implica asociar a cada arco e_{od} un *delta* $\delta_{od} \geq 0$, tal que $t_i = t + \delta_{od}$.

El arco e_{11} planifica el próximo arribo del evento v_1, que representa una nueva instancia de la tarea i. El retraso δ_{11} es calculado mediante:

$$\delta_{11} = \left\lfloor \frac{t + T_i}{T_i} \right\rfloor T_i - t$$

Los arcos e_{12} y e_{32} programan el evento v_2 en el instante actual, por lo tanto $\delta_{12} = \delta_{32} = 0$. El *delta* asociado al arco e_{23} es $\delta_{23} = C_i - c_i(t)$, el tiempo remanente de ejecución de la tarea. Luego, el evento v_3 es programado para el instante $t + \delta_{23}$. El arco e_{24} es recorrido cuando un arribo (evento v_1) interrumpe la ejecución de la instancia actual. El evento *desalojo* (v_4) será planificado en el mismo instante del evento v_1 más próximo (t_1). Luego el *delta* asociado a e_{24} será $\delta_{24} = t_1 - t$.

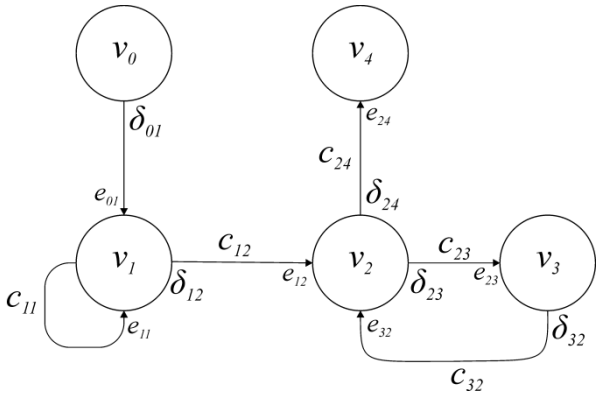

Figura 1. Grafo de eventos del modelo de un *STR*.

4.6. Ejecución de los Eventos

A continuación se detallan las modificaciones al estado del sistema (***S***) realizadas en cada evento. Esto es, la función f_i asociadas a cada evento v_i. En este modelo $\boldsymbol{S} = \boldsymbol{\Gamma}(n)$.

El evento *Inicialización* (v_0) programa las instancias de *Arribo* (v_1) iniciales, una por cada tarea del *STR* a simular. Se inicializa el reloj de la simulación, generalmente con $t = 0$. Cualquier otra actividad de *setup* debe de realizarse en este evento, como por ejemplo el cálculo de *tiempos ociosos en el sistema* en el instante crítico, *peores tiempos de respuesta,* etc.

El evento *Arribo* (v_1) agrega la tarea que recibió como parámetro en la cola de listos del planificador. Luego programa el arribo del próximo evento v_1 agregando la tupla $(t + \delta_{11}, v_1)$ a Λ. Finalmente, de cumplirse la condición c_{12}, se planifica un evento *Ejecución* (v_2) en el instante actual, $\Lambda \leftarrow (t, v_2)$.

El evento *Ejecución* (v_2), invoca el método o rutina correspondiente del planificador, para realizar las actividades relacionadas con la ejecución de la tarea de mayor prioridad en su cola de listos. Luego, si se satisface la condición c_{23} un evento *Finalización* (v_3) es planificado, $\Lambda \leftarrow (t + \delta_{23}, v_3)$. En caso de cumplirse c_{24}, un evento *Desalojo* (v_4) se programa para el instante $t + \delta_{24}$.

El evento *Finalización* (v_3) invoca los métodos o rutinas necesarios para actualizar ***S***, reflejando la finalización de la ejecución de la tarea de *TR* correspondiente. A continuación, de ser válida la condición c_{32}, un nuevo evento v_2 se programa para el instante actual, a fin de continuar ejecutando el resto de las tareas en la cola de listos del planificador.

De manera similar, el evento *Desalojo* (v_4) invoca los métodos o rutinas requeridas para modificar ***S*** debido al desalojo de una tarea del recurso.

Finalmente, el evento *Terminación* (v_5), programado en t_{end}, libera los recursos utilizados, e invoca las rutinas de generación de reportes necesarias.

5. Implementación

A continuación se ofrecerá una breve descripción de una implementación mediante la librería *SSJ* ([19]) para Java. Esta librería ofrece el paquete *simevents* para la simulación por eventos discretos. La misma cuenta con dos clases principales, *Simulator* y *Event*.

La clase *Simulator*, representa el ejecutivo de simulación. Provee el reloj de la simulación, la administración de la lista de eventos Λ, mediante diferentes implementaciones (lista doblemente enlazada, árbol binario, biselado, etc.) y métodos para iniciar, finalizar y reiniciar la ejecución de la simulación.

La clase *Event* representa una abstracción de un evento. Cada evento presentado en la sección 4.1 se implementa como una clase que extiende *Event* (*Init*, *Arrival*, *Run*, *End* y *Preempt*). Cada clase sobrecarga el método *actions*() con las acciones a realizar al ejecutar el evento, esto es las funciones f_v descritas en la sección 4.6.

Se asume que se cuenta con una lista o arreglo con el *STR* a ejecutar, y clases auxiliares que implementen, por ejemplo, el planificador (*Scheduler*) y los métodos o técnicas a evaluar.

La clase *Init* (v_0), en su método *actions*(), crea una instancia de *Arrival* (v_1) por cada tarea del *STR*. Estas instancias son programadas en los instantes correspondientes, mediante el método *schedule*(*delay*), provisto por la clase *Event*. La clase *Arrival*, desde su método *actions*(), agrega la instancia a la cola de listos del planificador. Luego comprueba la condición c_{12} y, de cumplirse, planifica una instancia de la clase *Run* (v_2). Esta clase invocará los métodos auxiliares necesarios para cualquier modificación en $\boldsymbol{S}$, y según se cumplan las condiciones c_{23} o c_{24}, programará una instancia de *End* (v_3) o *Preempt* (v_4) respectivamente. Tanto la clase *End* como *Preempt* invocarán métodos que implemente el planificador, según corresponda. Por ejemplo, *End* invocará el método *Scheduler.finishTask*(), y la clase *Preempt*, el método *Scheduler.preemptTask*(). De esta manera se desacopla de los eventos la lógica del planificador. La clase *End*, de cumplirse la condición c_{32}, programará una nueva instancia de la clase *Run*, en el instante actual. La clase *Simulator* ofrece métodos para inspeccionar Λ, a fin de comprobar las diferentes condiciones c_{od}.

6. Conclusiones y Trabajos Futuros

Se ha presentado un modelo de eventos discretos para la simulación de un *STR*, en base a un análisis de los *STR* como *sistemas dinámicos*, *discretos*, *no-lineales* y *determinísticos*. La técnica de *grafo de eventos* ha sido empleada para el diseño del modelo, debido a su sencillez y fácil implementación. El modelo resultante brinda una base teórica para el desarrollo de un simulador de *STR*, pudiendo utilizarse cualquiera de los paquetes *DES* disponibles en el mercado. A su vez, el presente trabajo sirve de base para futuros trabajos que extiendan el mismo, como por ejemplo para la simulación de *STR heterogéneos*.

7. Referencias

[1] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "STRESS: a simulator for hard real-time systems," *Softw. Pract. Exper.,* vol. 24, pp. 543-564, 1994.

[2] J. W. S. Liu, J. L. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. K. Shih, "PERTS: A prototyping environment for real-time systems," in *Real-Time Systems Symposium, 1993., Proceedings.*, 1993, pp. 184-188.

[3] F. Golatowski, J. Hildebrandt, J. Blumenthal, and D. Timmermann, "Framework for validation, test and analysis of real-time scheduling algorithms and scheduler implementations," in *Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on*, 2002, pp. 146-152.

[4] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," *Ada Lett.,* vol. XXIV, pp. 1-8, 2004.

[5] A. Diaz, R. Batista, and O. Castro, "Realtss: a real-time scheduling simulator," in *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, 2007, pp. 165-168.

[6] S. d. Vroey, J. Goossens, and C. Hernalsteen, "A Generic Simulator of Real-Time Scheduling Algorithms," presented at the Proceedings of the 29th Annual Simulation Symposium (SS '96), 1996.

[7] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano, "MAST: Modeling and analysis suite for real time applications," in *Real-Time Systems, 13th Euromicro Conference on, 2001.*, 2001, pp. 125-134.

[8] T. Kramp, M. Adrian, and R. Koster, "An Open Framework for Real-Time Scheduling Simulation," in *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, 2000, pp. 766-772.

[9] J. Teich, L. Thiele, and E. A. Lee, "Modeling and simulation of heterogeneous real-time systems based on a deterministic discrete event model," in *System Synthesis, 1995., Proceedings of the Eighth International Symposium on*, 1995, pp. 156-161.

[10] E. A. Lee, "Modeling concurrent real-time processes using discrete events," *Ann. Softw. Eng.,* vol. 7, pp. 25-45, 1999.

[11] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serius Problem for Next-Generations Systems," *IEEE Computer,* vol. Octubre, pp. 10-19, 1988.

[12] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM,* vol. 20, pp. 46-61, 1973.

[13] N. C. Audsley, A. Burns, M. F. Richarson, and A. J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach," in *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, GA, USA 1991.

[14] J. M. Urriza, R. Cayssials, and J. D. Orozco, "Modelado de Sistemas de Tiempo Real Planificados por RM o DM: Caracterización y Análisis," in *XXXIV Conferencia Latinoamericana de Informática, CLEI 2008*, Santa Fe, Argentina, 2008, pp. 1435-1444.

[15] M. Joseph and P. Pandya, "Finding Response Times in Real-Time System," *The Computer Journal (British Computer Society),* vol. 29, pp. 390-395, 1986.

[16] A. M. Law and W. D. Keaton, *Simulation Modelling and Analysis*, 2nd ed.: McGraw-Hill Higher Education, 1997.

[17] L. Schruben, "Simulation modeling with event graphs," *Commun. ACM,* vol. 26, pp. 957-963, 1983.
[18] E. L. Savage, L. W. Schruben, and E. Yücesan, "On the Generality of Event-Graph Models," *INFORMS J. on Computing,* vol. 17, pp. 3-9, 2005.
[19] P. L'Ecuyer and E. Buist, "Simulation in Java with SSJ," in *Simulation Conference, 2005 Proceedings of the Winter*, 2005, p. 10 pp.