Simulación de Tareas Aperiódicas y Esporádicas de Tiempo Real mediante un Modelo de Eventos Discretos

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

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

Resumen. Este trabajo presenta una extensión a un modelo de eventos discretos para el diseño de simuladores de Sistemas de Tiempo Real con tareas periódicas, incorporando tareas de tipo esporádico y aperiódico. Además, se estudia el soporte para *jitter* y *offset*. El modelo de eventos discretos elegido se ajusta a los sistemas dinámicos discretos como lo son los Sistemas de Tiempo Real. El uso de simuladores para realizar comprobaciones de algoritmos, modelos, técnicas y para evaluar métricas de rendimiento, es una práctica extendida e importante en la disciplina. El modelo, es diseñado empleando la técnica de grafo de eventos.

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

1 Introducción

La simulación por computadora es, en la actualidad, una herramienta esencial en un gran número de disciplinas. Es una herramienta importante para la obtención de nuevos resultados, acelerando el proceso de desarrollo de nuevos métodos y técnicas. Sin embargo, pocas veces se hace énfasis en la revisión y validación del *software* empleado para la obtención de los resultados publicados. Es evidente la importancia de validar dichos resultados, mediante el uso de soluciones *software* similares, por parte de otros investigadores.

Un modelo de eventos discretos para la simulación de Sistemas de Tiempo Real (*STR*) es presentado en [1]. Dicho modelo sirve como base sobre la cual desarrollar sistemas de simulación por computadora, y ha sido empleado con éxito en el simulador de *STR* del grupo de investigación de la UNPSJB[1]. En este trabajo se extiende dicho modelo, agregando soporte para tareas de aperiódicas y esporádicas, y exten-

[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).

diendo el modelo de tareas periódicas con soporte para *jitter* de instanciación y *offsets*.

En la actualidad, un conjunto de aplicaciones y *frameworks* existen en la literatura para la simulación de *STR*. Entre los más relevantes se pueden mencionar a STRESS ([2]), PERTS ([3]), YASA ([4]), Cheddar ([5]), RealTTS ([6]) y el simulador de la Université Libre de Bruxelles ([7]). Aplicaciones como MAST ([8]) ofrecen herramientas de modelado, y otras como FORTISSIMO ([9]) presentan una plataforma para el diseño de simulaciones. Un estudio de los *STR* como *sistemas discretos* para lograr análisis temporales eficientes se encuentra en [10], en donde son modelados mediante redes de Petri. En [11] se presenta un marco general para el estudio de *STR* como sistemas discretos.

El presente trabajo se encuentra organizado de la siguiente manera: en la sección 2 se realiza una introducción a los *STR*. La sección 3 expone la simulación por eventos discretos y el modelado por grafo de eventos. Luego, en la sección 4, se realiza un resumen del modelo presentado en [1]. A continuación, las secciones 5 y 6 presentan las extensiones necesarias para el modelado de tareas aperiódicas y esporádicas. Las secciones 7, 8 y 9 describen la ejecución del modelo y una implementación de referencia. Finalmente, la sección 10 expone las conclusiones y posibles trabajos futuros.

2 Introducción a los *STR*

Una definición de *STR* ampliamente aceptada en la disciplina, es la formulada por Stankovic en [12]. La misma 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*".

De acuerdo al vencimiento, los *STR* se clasifican como *duros*, *blandos* o *firmes*. Un *STR* tipo *duro* no tolera la perdida de ningún vencimiento bajo ninguna circunstancia. Un *STR* tipo *blando* puede permitir la perdida de algunos vencimientos. Finalmente, un *STR* tipo *firme* tipifica las pérdidas según algún criterio estadístico.

El presente trabajo utiliza un modelo de *STR* mono recurso, con tareas independientes y apropiables. Una tarea i (τ_i) periódica de tiempo real es caracterizada por su peor tiempo de ejecución (C_i), periodo (T_i), vencimiento (D_i), *jitter* de instanciación (J_i) y *offset* (Of_i). Cada τ_i genera una secuencia de trabajos o instancias, siendo $j_{i,k}$, la k-ésima instancia. El tiempo ejecutado por una instancia $j_{i,k}$ al instante t, se representa como $c_{i,k}(t)$. Se considera que las tareas periódicas finalizan antes del arribo de una nueva instancia. Consecuentemente, no existen instancias previas de una misma tarea periódica esperando ser ejecutadas. Luego, un conjunto de n tareas periódicas de Tiempo Real se define como $\Gamma(n) = \{(C_1, T_1, D_1, J_1, Of_1), ..., (C_n, T_n, D_n, J_n, Of_n)\}$.

En [13] se demostró que el peor estado de carga, para un planificador mono-recurso, es aquel instante en que todas las tareas solicitan ejecución simultáneamente, y se lo denomino *instante crítico*. Si el planificador puede ejecutar todas las instancias de las tareas que arribaron en el *instante crítico* sin perder vencimientos, se considera al *STR planificable*. El algoritmo de planificación puede realizar una asignación está-

tica sobre el recurso compartido, o basada en prioridades (*Rate Monotonic, RM* [13] o *Deadline Monotonic, DM* [14]).

En lo que sigue, se denominarán a las tareas que cuentan con tiempos de arribo arbitrarios, como *tareas aperiódicas*, siendo del tipo *blando*. Por otro lado, a las tareas con arribos arbitrarios y con requerimientos *duros* se las denominarán *tareas esporádicas*. Para cumplir con el requerimiento temporal duro, estas últimas cuentan con un intervalo mínimo entre arribos [15].

En la literatura existen varias técnicas y métodos para la atención de estos tipos de tareas, por ejemplo buscando minimizar su tiempo de respuesta, garantizando al mismo tiempo el cumplimiento temporal de las tareas periódicas de tipo *duro*.

En [16] se estudió la caracterización de los *STR* como *sistemas dinámicos*, *no-lineales*, *discretos* y *determinísticos* al ser diagramados por medio de disciplinas de prioridades fijas, tales como *RM* o *DM*.

3 Simulación por Eventos Discretos

En esta sección se presenta una breve introducción a la *simulación por eventos discretos*, y a la técnica de modelado *grafo de eventos*.

La *simulación por eventos discretos* se aplica en el estudio de sistemas que pueden ser representados con modelos lógico-matemáticos discretos. En estos, las variables son modificadas en un determinado número finito de instantes, dentro de un intervalo determinado de tiempo ([17]). El conjunto atómico de modificaciones sobre las variables de estado del sistema, en un instante determinado, se denominado *evento* (v).

En la simulación se emplea un *reloj*, t, que registra el tiempo actual, y una lista de eventos futuros Λ. La lista Λ, es un conjunto de tuplas (t_i, v_i), siendo $t_i \geq t$ el instante en donde el evento v_i será ejecutado. La misma es implementada como una cola de prioridades ordenada según los valores t_i. La simulación consiste en remover la primer tupla (t_i, v_i) en Λ, actualizar t con el valor t_i, y luego ejecutar las acciones asociadas al evento v_i. Una vez terminadas las acciones, se continúa con la siguiente tupla. Los procesos concurrentes pueden ser simulados mediante múltiples eventos planificados en un mismo t_i. La simulación cuenta con un conjunto de eventos iniciales, generalmente programados para $t = 0$. La simulación finaliza cuando t supera un valor determinado (t_{end}), se encuentra un evento de finalización (v_{end}) o Λ no contiene más elementos ($\Lambda = \varnothing$).

3.1 Grafo de Eventos

La técnica de *grafo de eventos* ([18, 19]) permite formalizar un modelo de eventos discretos. Los eventos identificados en el modelo son representados mediantes nodos, y las relaciones lógica-temporales entre estos son indicadas mediante arcos. Notar que el grafo de eventos *no representa un autómata*. Un *grafo de eventos* $\boldsymbol{M}$ consiste de:

- $\boldsymbol{S}$, el conjunto de variables y atributos que conforman el estado del sistema.
- $\boldsymbol{V}$, un conjunto de nodos que representan los eventos identificados.

- $\boldsymbol{E}$, un conjunto de arcos dirigidos $e_{od} = (v_o, v_d)$, que representan la relación lógica-temporal, entre un *evento origen* (v_o) y otro *destino* (v_d).
- $\boldsymbol{F} = \{f_v : \boldsymbol{S} \to \boldsymbol{S} \; \forall \; v \in \boldsymbol{V}\}$, funciones asociadas a cada nodo $v \in \boldsymbol{V}$, que describen los cambios en $\boldsymbol{S}$, producto de la ejecución de un evento.
- $\boldsymbol{C} = \{c_{od} : \boldsymbol{S} \to \{0,1\} \forall e_{od} \in \boldsymbol{E}\}$, condiciones asociadas a cada arco e_{od}.
- $\boldsymbol{D} = \{\delta_{od} \in \mathbb{R}_0^+ \forall e_{od} \in \boldsymbol{E}\}$ *deltas* asociados a cada arco e_{od}.
- $\boldsymbol{A} = \{\boldsymbol{A}_e, e_{od} \in E\}$, atributos asociados al arco e_{od} y $\boldsymbol{B} = \{\boldsymbol{B}_v, v \in V\}$, parámetros asociados al evento v.

El modelo es especificado como $\boldsymbol{M} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{F}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{B})$. Cada arco e_{od} se recorre si la condición c_{od} es válida luego de ejecutar el evento v_o. Esto equivale a programar el evento v_d en el instante $t + \delta_{od}$ (δ_{od} es el *delta* asociado a e_{od}). El conjunto de atributos $\boldsymbol{A_e}$ del arco e_{od}, son los argumentos que el evento v_d espera recibir como parámetros ($\boldsymbol{B_v}$), y son opcionales. Tanto Λ como t están asociados con la ejecución de la simulación de $\boldsymbol{M}$, y no son partes del modelo. Esta técnica es empleada en [1], para realizar un modelo de *STR*, que es resumido a continuación.

4 Modelo de eventos discretos para un *STR* de tareas periódicas

A continuación se resume el modelo de eventos discretos desarrollado en [1]. Este modelo identifica seis eventos asociados a tareas periódicas. Estos son:

- *Inicio* (v_0, prioridad 0): Corresponde al inicio de la simulación.
- *Terminación* (v_{end}, prioridad 1): Indica la finalización de la simulación.
- *Arribo* (v_1, prioridad 4): Arribo de una nueva instancia de una tarea periódica.
- *Ejecución* (v_2, prioridad 5): Ejecución de la instancia con mayor prioridad.
- *Finalización* (v_3, prioridad 2): Finaliza la ejecución de una instancia.
- *Desalojo* (v_4, prioridad 3): Posible desalojo de una instancia del recurso.

El evento *Arribo* recibe como parámetro la nueva instancia $j_{i,k}$ de una τ_i. Luego, habrá tantos eventos v_1 como tareas tenga el *STR*. Las prioridades asignadas permiten ordenar correctamente la ejecución de eventos planificados para un mismo instante t_i. La máxima prioridad es cero. Las relaciones entre estos eventos se especifican mediante los siguientes arcos:

- e_{01}: planifica el arribo de las primeras instancias de las tareas periódicas.
- e_{11}: programa el arribo de la próxima instancia de una tarea periódica (evento v_1).
- e_{12}: planifica la ejecución de la instancia de mayor prioridad (evento v_2).
- e_{23}: programa la finalización de la instancia en ejecución (evento v_3).
- e_{24}: planifica el posible desalojo de la instancia en ejecución (evento v_4).
- e_{32}: programa la ejecución de la instancia de mayor prioridad (evento v_2).

Cada arco tiene asociada una condición, que de cumplirse indica que se debe programar el evento en el otro extremo del mismo:

- c_{11}: La instancia previa de la tarea τ_i no excedió su tiempo de ejecución.

- c_{12}: No existen otros eventos de tipo v_1 en Λ para el instante actual.
- c_{23}: La instancia finaliza su ejecución antes del arribo del próximo evento v_1.
- c_{24}: La instancia actual no finaliza antes del arribo del próximo evento v_1.
- c_{32}: No existen eventos de tipo v_1 en Λ para el instante actual, y existe al menos una instancia sin finalizar en la cola de listos del planificador.

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

Si una condición se evalúa positivamente, el evento destino del arco al que dicha condición corresponde, se programa para un instante $t_i \geq t$. El modelo de *grafo de eventos* especifica dicho instante mediante un *delta* $\delta_{od} \geq 0$, uno por arco, tal que $t_i = t + \delta_{od}$. Luego, los *deltas* del modelo son:

- $\delta_{11} = \lfloor t + T_i / T_i \rfloor T_i - t$, siendo $\lfloor t + T_i / T_i \rfloor T_i$ el instante donde ocurre el próximo arribo de una instancia de la tarea τ_i.
- $\delta_{23} = C_i - c_i(t)$, que es el tiempo remanente de ejecución de la instancia de mayor prioridad en la cola de listos del planificador.
- $\delta_{24} = t_1 - t$, siendo t_1 el instante donde está programado el evento v_1 más próximo.
- $\delta_{01} = \delta_{12} = \delta_{32} = 0$, ya que el evento destino se planifica para el instante actual.

El arco e_{24} se recorre cuando un evento *Arribo* (v_1) interrumpe la ejecución de la instancia actual. Por lo tanto el evento *Desalojo* (v_4) es planificado en el tiempo de arribo de dicho evento, t_1, y se tiene que $\delta_{24} = t_1 - t$.

5 Modelado de tareas aperiódicas y esporádicas

Existen dos opciones para agregar las tareas aperiódicas y esporádicas al modelo. Una primera opción es modificar la lógica del evento *Arribo* para que administre estos tipos de tareas, en conjunto con las tareas periódicas. Esto evita incrementar el número de eventos en el modelo. Como contrapartida, la lógica del evento *Arribo* se torna más compleja. Una segunda opción es agregar nuevos eventos para el arribo de tareas aperiódicas y esporádicas. Esto incrementa la complejidad del modelo, pero permite un mayor detalle y control. En este trabajo se empleará esta última alternativa. Los siguientes eventos son agregados:

- *ArriboA* (v_6, prioridad 4): Arribo de una nueva instancia de una tarea *aperiódica*.
- *ArriboE* (v_7, prioridad 4): Arribo de una nueva instancia de una tarea *esporádica*.

Ambos eventos, tienen como parámetro la instancia de τ_i que debe arribar. El evento *Arribo* (v_1) destinado a las tareas periódicas, se renombra como *ArriboP*. Se denominará a los eventos v_1, v_6 y v_7 como eventos de tipo *Arribo*. Los nuevos arcos son:

- e_{06} y e_{07}, que programan el primer evento de v_6 y v_7 en la simulación.
- e_{66} y e_{77}, que planifican el próximo arribo del evento v_6 o v_7, respectivamente.
- e_{62} y e_{72}, planifica la ejecución de la instancia con mayor prioridad (evento v_2).

Las condiciones asociadas a los nuevos arcos son:

- c_{77}: La instancia previa de la tarea esporádica no excedió su tiempo de ejecución.
- c_{72} y c_{62}, ambas verifican si no existen otros eventos de tipo *Arribo* en Λ para el instante actual.

Los *deltas* asociados con los nuevos arcos son:

- δ_{66}, es el próximo tiempo de arribo de la instancia de la tarea aperiódica.
- δ_{77}, es el próximo tiempo de arribo de la instancia de la tarea esporádica, debe respectar el tiempo mínimo entre arribos de la tarea.
- $\delta_{72} = \delta_{62} = 0$, ya que el evento v_2 se planifica para el instante actual.

Los eventos v_6 y v_7 agregan la instancia de la tarea aperiódica o esporádica que recibieron como parámetro al planificador. Como la ejecución de estas tareas (por medio de servidores, uso de *slack*, o relegación a ejecución en *background*) es decisión del planificador (que es activado por un evento v_2), y no del modelo de eventos, el orden en que son procesados los eventos de tipo *Arribo* (v_1, v_6 o v_7) programados en un mismo instante, no es determinante. Luego, todos cuentan con la misma prioridad (4). Como las instancias de una tarea aperiódica pueden acumularse no es necesario una condición asociada al arco e_{66} que controle la programación de nuevos arribos.

6 Jitter y Offset

El *Offset* de una tarea τ_i (Of_i) es el máximo desplazamiento que sufre desde su tiempo de arribo, respecto del *instante crítico* ([20, 21]). Es útil, por ejemplo, para modelar relaciones de precedencia, evitando que una tarea sea instanciada antes que otra haya terminado su ejecución. Debido a que es una perturbación que afecta a todas las instancias de una tarea, se puede modelar sumando el valor del *offset* al *delta* utilizado para la planificación del evento *ArriboP*. Luego el *delta* será $\delta_{11} + Of_i$.

El *jitter de instanciación* (J_i), en su caso más común, es el tiempo insumido por el *Sistema Operativo de Tiempo Real* en tener lista para ejecución la tarea τ_i (desde su arribo). Este tiempo es acotado y no afecta al período de la tarea (T_i). La presencia de *jitter* en una tarea puede generar una mayor interferencia en las tareas de menor prioridad, lo que puede resultar en pérdidas de vencimientos de las mismas. Estos casos son estudiados en [22, 23, 24, 25], entre otros.

Para modelar el efecto del *jitter* sobre las instancias, se tendrá en cuenta el hecho que, hasta que el *jitter* no haya expirado, la instancia no será visible para el planificador. De esta manera, antes de programar el evento *ArriboP*, para el arribo de la próxima instancia de τ_i, se calcula el valor de *jitter* de dicha instancia. Luego, *ArriboP* será programado en el instante $t + \delta_{11} + J_i$. Luego, para agregar el soporte para *jitter* y *offset*, se debe modificar el delta asociado al arco e_{11}, siendo su nuevo valor $\delta_{11} = \lfloor t + T_i / T_i \rfloor T_i + J_i + Of_i - t$.

A continuación se presenta el grafo para el nuevo modelo de eventos discretos. A fin de simplificar el diagrama, se omite el evento *Terminación*, así como los *deltas* con un valor igual a cero. Los arcos e_{11}, e_{66} y e_{77} también se omiten por claridad.

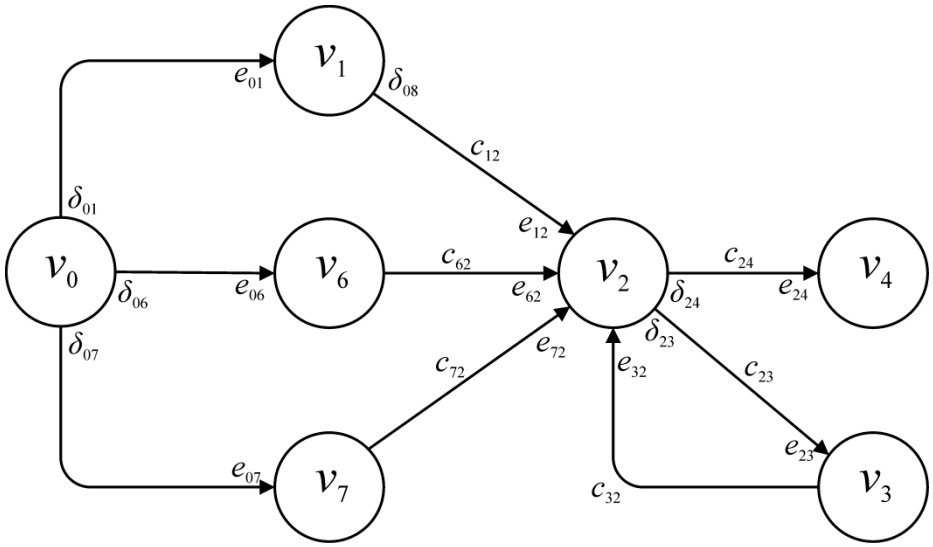

Figura 1. Grafo de Eventos parcial para el *STR*.

7 Análisis de la Cola de Eventos Futuros

La cola de eventos futuros (Λ) contiene los próximos eventos a ejecutarse. De acuerdo a los arcos y condiciones entre los eventos v_2, v_3 y v_4, se puede observar que para cualquier instante, existe en Λ una única instancia de v_2, v_3 o v_4.

Consideremos un evento *ArriboP* (v_1). La ejecución de un evento de este tipo, programa en Λ su próxima instancia. De esta manera, y recordando que se remueve el evento de la lista antes de su ejecución, existe en Λ a lo sumo una instancia de v_1. Como cada evento *ArriboP* representa una tarea en particular del *STR* simulado, para un *STR* con n tareas, habrá como máximo n eventos v_1 en Λ en cualquier instante.

La situación para los eventos de tipo *ArriboE* (v_7) es análoga. Luego, dado un *STR* con n tareas periódicas y m tareas esporádicas, habrá a lo sumo $n + m$ eventos v_2 y v_7, en Λ.

La situación para el evento *ArriboA* es similar. El próximo evento v_6 puede ser generado por la ejecución del evento anterior. O bien todos los eventos *ArriboA* esperados pueden ser programados al inicio de la simulación.

Notar también que, durante la ejecución de un evento v_2, v_3 o v_4, el próximo evento en Λ es siempre de tipo *Arribo* (v_1, v_6 o v_7). Luego, se pueden simplificar las condiciones c_{12}, c_{23}, c_{24} y c_{32}, sustituyendo v_1 por el tiempo de ocurrencia del siguiente evento en Λ.

8 Ejecución del modelo

A continuación se describe genéricamente la ejecución del modelo, y las acciones a realizar por parte de los eventos.

El evento *Inicialización* (v_0) crea, por cada tarea del *STR*, un evento del tipo *Arribo* (v_1, v_6 o v_7) apropiado. Estos son los eventos iniciales de la simulación. Se inicializa el reloj, generalmente en $t = 0$. Otras actividades de *setup* se pueden realizar en v_0 (por ejemplo, cálculo de *slack* en el instante crítico, *peores tiempos de respuesta*, etc.). Luego, la simulación toma el primer evento en Λ, y lo ejecuta.

La ejecución de un evento de tipo *Arribo* (v_1, v_6 o v_7) agrega una nueva instancia de una tarea de tiempo real al planificador, y programa su próxima ejecución en Λ, en un instante futuro, de acuerdo al *delta* correspondiente. Luego, de cumplirse alguna de las condiciones c_{12}, c_{62}, o c_{72} (según el tipo de evento *Arribo* ejecutado) un evento *Ejecución* (v_2) es planificado en el instante actual.

El evento *Ejecución* (v_2), invoca la rutina del planificador a fin de simular la ejecución de la instancia con mayor prioridad en la cola de listos. Luego, si se satisface la condición c_{23}, un evento *Finalización* (v_3) es planificado en $t + \delta_{23}$. Caso contrario la condición c_{24} será válida, y un evento *Desalojo* (v_4) se programa para $t + \delta_{24}$.

El evento *Finalización* (v_3) invoca las rutinas del planificador que simulan el fin de ejecución de una instancia. A continuación, de ser válida la condición c_{32}, un nuevo evento v_2 se programa en el instante actual. De esta manera se continúa ejecutando las tareas de la cola de listos del planificador. De manera similar, el evento *Desalojo* (v_4) invoca las rutinas requeridas para realizar el desalojo de una tarea. Como el evento *Desalojo* se programa en el mismo instante que un evento de tipo *Arribo*, este último se encargará de generar un evento v_2.

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

Se hace notar, que el planificador se ejecuta únicamente a pedido de los distintos eventos. Por lo tanto, su única responsabilidad es mantener la cola de tareas listas, y decidir que tarea debe ejecutar según la política de planificabilidad simulada.

9 Implementación

A continuación se ofrecerá una síntesis de una implementación mediante la librería *SSJ* ([26]) para Java. Esta librería ofrece dos clases principales para la simulación por eventos discretos, *Simulator* y *Event*.

La clase *Simulator* representa el ejecutivo de simulación. Administra el reloj, la lista de eventos Λ, y ofrece métodos para iniciar, finalizar y reiniciar la simulación. Cada evento del modelo se implementa como una clase que extiende *Event* (*Init*, *ArrivalP*, *ArrivalE*, *ArrivalA*, *Run*, *End* y *Preempt*), y sobrecarga el método *actions*() con las acciones a realizar al ejecutar el evento. La clase *Event* es una clase abstracta que representa a un evento genérico.

Se presume que existe una estructura de datos con el *STR* a simular, y clases auxiliares que implementan el planificador (*Scheduler*) y los métodos o técnicas a evaluar.

El método *actions*() de la clase *Init* (v_0), crea las instancias iniciales de *ArrivalP* (v_1), *ArrivalA* (v_6) y *ArrivalE* (v_7). Estas instancias son programadas mediante el método *schedule*(*delay*), de la clase *Event*. Las clases *Arrival*, desde el método *actions*(), agregan la nueva instancia de una tarea τ_i al planificador. Cada clase comprueba su condición c_{od} asociada, y de cumplirse, planifica una nueva instancia de *Run* (v_2) en el instante actual.

La clase *Run* invocará, por ejemplo, *Scheduler.runTask*(), que simulará la ejecución de la instancia de mayor prioridad. Luego, según se cumplan las condiciones c_{23} o c_{24}, se programará una instancia *End* (v_3) o *Preempt* (v_4), que invocarán los métodos necesarios en el planificador, según corresponda. Por ejemplo, *End* invocará *Scheduler.finishTask*(), y *Preempt* el método *Scheduler.preemptTask*(). Así la lógica del planificador se desacopla de la de los eventos. La clase *End*, de cumplirse la condición c_{32}, planifica una nueva instancia de la clase *Run*, para el instante actual. La clase *Simulator* ofrece métodos para inspeccionar Λ, a fin de comprobar las diferentes condiciones c_{od}.

10 Conclusiones y Trabajos Futuros

Se ha presentado una extensión al modelo de eventos discretos presentado en [1], agregando soporte para la simulación de tareas aperiódicas y esporádicas, así como también para los casos de *jitter* y *offset* de las tarea periódicas de tiempo real.

El modelo resultante brinda una base formal para el desarrollo de un simulador de *STR*, pudiendo hacerse uso de cualquiera de los paquetes DES disponibles en la actualidad. A su vez, el presente trabajo sirve de base para futuras extensiones del mismo, como por ejemplo para la simulación de *STR heterogéneos*.

11 Referencias

[1] F. E. Paez, J. M. Urriza, C. E. Buckle, S. Lucas, and J. D. Orozco, "A Discrete Event Model for Real Time System Simulation," *Journal of Computer Science and Technology (JCS&T),* vol. 12, pp. 99-103, 2012.

[2] 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.

[3] J. W. S. Liu, J. L. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman*, et al.*, "PERTS: A prototyping environment for real-time systems," in *Real-Time Systems Symposium, 1993., Proceedings.*, 1993, pp. 184-188.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] B. Sprunt, "Aperiodic Task Scheduling for Real-Time Systems," Doctor in Computer Engineering, Department of Electrical and Computer Engineering, Carnegie Mellon University, Carnegie Mellon University, 1990.

[16] 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.

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

[18] L. Schruben, "Simulation modeling with event graphs," *Commun. ACM,* vol. 26, pp. 957-963, 1983.

[19] 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.

[20] J. C. Palencia and M. Gonzalez Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, 1998, pp. 26-37.

[21] I. Bate and A. Burns, "Schedulability analysis of fixed priority real-time systems with offsets," in *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, 1997, pp. 153-160.

[22] K. W. Tindell, "Fixed Priority Scheduling of Hard Real-Time Systems," Doctor of Philosophy, Department of Computer Science, University of York, 1993.

[23] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings, "Applying New Scheduling Theory to Static Priority Preemptive Scheduling," *Software Engineering Journal,* vol. 8, pp. 284-292, 1993.

[24] P. Richard and J. Goossens, "Approximating Response Times of Static-Priority Tasks with Release Jitters," in *Euromicro Conference on Real-Time Systems. WIP* Dresden, Germany, 2006, p. 4.

[25] O. Redell and M. Torngren, "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter," in *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, 2002, pp. 164-172.

[26] P. L'Ecuyer and E. Buist, "Simulation in Java with SSJ," in *Simulation Conference, 2005 Proceedings of the Winter*, 2005, p. 10 pp.