A Discrete Event Model for Real Time System Simulation

Francisco E. Paez, José M. Urriza, Carlos E. Buckle, Lucas Schorb
Facultad de Ingeniería – Depto. de Informática
Universidad Nacional de la Patagonia
Puerto Madryn, Argentina
e-mail: josemurriza@gmail.com

Javier D. Orozco
Depto. de Ingeniería Eléctrica y Computadoras
Universidad Nacional del Sur - CONICET
Bahía Blanca, Argentina
e-mail: jadorozco@gmail.com

***Abstract* — In this work we present a discrete event model to design and implement a real time system simulator. This kind of software is useful to verify and evaluate algorithms and models, and to compute performance metrics. The discrete event model fits perfectly with discrete dynamical system such as Real Time Systems. The event graph technique is then used as the modeling tool.**

Keywords- Real Time Systems simulation; Modeling; Discrete Event Systems.

I. INTRODUCTION

Computer aided simulation is an essential tool in a large number of disciplines. It plays a key role accelerating the creation process of investigation methods and technics. However, the review and validation of simulation software and the techniques used to get these investigation results are often an overlooked issue. It is important that other research groups can validate their results by reproducing experiments using the same simulation software. Or at least with one that uses the same or similar model.

The objective of this work is to formulate a discrete event model for developing Real Time Systems (*RTS*) simulation software. The model presented was used as basis for our investigation group[1] *RTS* simulation software.

In the past several applications have been developed for *RTS* simulation: STRESS ([1]), PERTS ([2]), YASA ([3]), Cheddar ([4]), RealTTS ([5]) and the Université Libre de Bruxelles simulator [6], to name a few. Some development modeling tools are MAST ([7]) and FORTISSIMO ([8]). Works that studies *RTS* as discrete systems are [9] and [10].

In [10] is presented a general framework for studying this kind of systems.

This paper is organized as follows: Section 2 presents an introduction to *RTS*, the task model and the notation. Section 3 presents an introduction to the discrete event modeling and simulation, and an overview of the *event graph* technique. In Section 4 the model development is presented. Section 5 discusses a reference implementation. Finally, Section 6 presents our conclusion and future work.

II. REAL TIME SYSTEMS

Stankovic presented in [11] a formal definition accepted by the community discipline: "*In real-time computing the correctness of the system depends not only in the logical result of the computation but also on the time at which the results are produced*".

Depending on how critical it is to meet the deadline, a *RTS* can be classified as a *hard*, *soft* or *firm* one. A hard *RTS* does not tolerate any deadline loss. In contrast, a soft *RTS* can afford to lose some deadlines. Finally a firm *RTS* typify the losses according to some statistical criterion.

This work uses the single-processor, multiprogrammed system model presented in [12]; the tasks are periodic, preemptable and independent of each other. A scheduling algorithm is used to determine which task has to be executed at a particular instant. This algorithm could perform a *static* assignation over the shared resource or an assignation based on priorities.

Under this model, a real time task i (τ_i) is characterized by its worst case execution time (C_i), period (T_i) and deadline (D_i). A set of n real time tasks is then specified as $\Gamma(n) = \{(C_1, D_1, T_1), \ldots, (C_n, T_n, D_n)\}$. Each task generates an infinite sequence of jobs (instances), where $j_{i,k}$ denotes the kth job of a task τ_i. The executed time of a job $j_{i,k}$ at a time t is denoted as $c_{i,k}(t)$.

Also, in [12] was proved that a single-processor scheduler worst state of load occurs when all jobs require execution simultaneously. This instant is known as *critical instant*. If all the jobs can execute without missing its deadlines from this instant, then it is said that the *RTS* is schedulable.

A. Characterization and Analysis of a Real Time System

A *dynamical system* is one whose state changes in function of time. Then a *RTS* is such a system. A study of *RTS* as *dynamical systems*, when scheduled by *Rate Monotonic* (*RM*, [12]) or *Deadline Monotonic* (*DM*, [13]) algorithms, can be found in [14]. In this work the *RTS* evolution since the *critical instant* is modeled using a fixed point (*FP*) equation [15] which calculates a task worst response time:

[1]Real Time Systems Group - Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB) Sede Puerto Madryn (http://www.rtsg.unp.edu.ar).

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

This *FP* equation models the evolution of the $\mathbf{\Gamma}(i)$ subsystem starting at the *critical instant* by using an iterative process. The method stops when finds a *FP* such that $t^q = t^{q+1} \leq D_i$, in which case the system is schedulable. If the *FP* is found after the deadline, the system is not schedulable ($t^{q+1} > D_i$). For certain systems such *FP* may not exist. The equation (1) is monotonic, deterministic (for each value of t there exists only one result), non-linear and describes a dynamic system.

Therefore, a *RTS* could be characterized as a dynamic, non-linear, discrete and deterministic system ([14]) when it is scheduled by a fixed priority scheduler, such as *RM* or *DM*.

III. Discrete Event Simulation

Following there is a brief introduction to *discrete event simulation* (*DES*) and the *event graph* technique is presented.

Discrete event simulation is used to study and simulate systems which can be represented by discrete models[2]. In such model, given a finite interval of time, state variables changes instantaneously only in a certain number of moments ([16]). An event (v) is defined then as an atomic set of changes over the state variables at a certain time.

The models are generally represented as recursive relationships, for example $t^{k+1} = 2t^k$, where k denotes the number of discrete steps. Equation (1) is also an example of such model.

During simulations a clock time t is maintained with the actual simulation time and a *future event list* Λ. This list is a collection of (t_i, v_i) tuples, being $t_i \geq t$ the instant in which the event v_i will be executed. Generally Λ is a priority queue prioritized according to the values of t_i.

At each step of the simulation, the first event in Λ (the one with the maximum priority) is executed and t is updated with the value of t_i. Concurrent processes can be simulated scheduling multiple events at the same time. The simulation starts with a set of events usually scheduled at $t = 0$. It ends when one of these situations occurs: $\Lambda = \varnothing$ (there are no further futures events); $t \geq t_{end}$, where t_{end} is a predetermined ending time; or a specific termination event (v_{end}) is executed.

A. Event Graphs

A discrete event model can be developed with the *Event Graphs* technique ([17, 18]). The dynamics of the modeled system are represented with *events* which depict system's state changes. The logical and temporal relationships between them are indicated by *edges*. It is important to note that the final *event graph* it is not an automaton.

An *event graph* model $\boldsymbol{M}$ has the following components:

- $\boldsymbol{S}$, the set of variables that conforms the system state.
- $\boldsymbol{V}$, a set of vertexes, each corresponding to one event.
- $\boldsymbol{E}$, the set of directed edges $e_{od} = (v_o, v_d)$ that describes the scheduling relationship between two events v_o and v_d in $\boldsymbol{V}$.
- $\boldsymbol{F} = \{f_v : \boldsymbol{S} \to \boldsymbol{S} \ \forall\ v \in \boldsymbol{V}\}$, the state changes functions associated with each vertex $v \in \boldsymbol{V}$. They describe the state changes on $\boldsymbol{S}$ when an event v executes.
- $\boldsymbol{C} = \{c_{od} : \boldsymbol{S} \to \{0,1\} \forall e_{od} \in \boldsymbol{E}\}$, edge condition functions associated with each *edge* e_{od}. The *edge* e_{od} is traversed if and only if $c_{od} = 1$.
- $\boldsymbol{D} = \{\delta_{od} \in \mathbb{R}_0^+ \forall e_{od} \in \boldsymbol{E}\}$, the set of time delays. One for each edge e_{od}.
- $\boldsymbol{A} = \{\boldsymbol{A_e}, e_{od} \in E\}$, the set of attributes, if any, associated with each edge e_{od}.
- $\boldsymbol{B} = \{\boldsymbol{B_v}, v \in V\}$, the set of parameters, if any, associated with each vertex v.

Then an *event graph* model is specified as the set $\boldsymbol{M} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{F}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{B})$. Each directed edge $e_{od} = (v_o, v_d)$ is traversed if and only if the associated edge condition c_{od} evaluation is true after the execution of the event v_o. To traverse an edge e_{od} means to schedule an event v_d at the instant $t + \delta_{od}$, where δ_{od} is the time delay of the edge e_{od}. The set of state variables modified by f_v is known as $\boldsymbol{S_v}$, $\boldsymbol{S_v} \subseteq \boldsymbol{S}$.

Given an edge e_{od}, the associated set of attributes $\boldsymbol{A_e}$ will be the formal arguments required by event v_d (set $\boldsymbol{B_v}$). If no parameters are needed, then $\boldsymbol{A}$ and $\boldsymbol{B}$ are empty sets.

It is important to note that Λ and t (the simulation clock), are associated with a simulation execution of the model. Therefore are not themselves part of $\boldsymbol{M}$.

The event graph model technique will be used in the next section to model a *RTS* as a discrete event system.

IV. A Discrete Event Model Of a RTS

Following a discrete event model for a *RTS* is developed with the event graph technique. The model identifies the instantiation of new jobs, and schedule events for the execution, finalization and preemption of these jobs.

A. System state

The system state $\boldsymbol{S}$ is composed by a set of n real-time tasks, $\mathbf{\Gamma}(n)$, and the most recent job $j_{i,k}$ for each task. The jobs are grouped in a *ready queue*, sorted according to the task priorities.

B. Events

The model identifies six different events. The first event, v_0, corresponds to the *RTS setup time*. For any job $j_{i,k}$ the following events are identified: *Arrival* (v_1), *Execution* (v_2), *Finalization* (v_3) and *Preemption* (v_4). The v_1 event receives the job $j_{i,k}$ as a parameter. Finally, an event *EndSimulation* (v_5) is scheduled at time t_{end} when the simulation should end.

C. Simultaneous Event Precedence

Two or more events might be scheduled at the same simulated time t. An erroneous execution order of these simultaneous events could result in an invalid state of $\boldsymbol{S}$. An appropriate execution priority assignation to each event type helps to solve this problem. The priorities assignment is showed in Table 1. The maximum priority is 0.

---

[2] The system under study could be either *discrete* or *continuous*.

TABLE I
EVENT EXECUTION PRIORITIES

Event	Priority
Initialization (v_0)	0
EndSimulation (v_5)	1
Finalization (v_3)	2
Preemption (v_4)	3
Arrival (v_1)	4
Execution (v_2)	5

D. Edges

The events v_0 through v_4 are connected by six edges, as shown in Figure 1. The edges are:

- e_{01}: Schedule the arrival of the first job of each task ($j_{i,1}$ for $i = 1$ .. n).
- e_{11}: Schedule the next event v_1 (*Arrival*).
- e_{12}: Schedule a new event v_2 (*Execution*).
- e_{23}: Schedule a new event v_3 (*Finalization*).
- e_{24}: Schedule a new event v_4 (*Preemption*).
- e_{32}: Schedule a new event v_2 (*Execution*).

E. Edge condition functions

The edge e_{01} schedules the first job of each task. Then it is traversed only at the simulation's start. In this work it hasn't got an associated edge condition, but one could be added to, for example, perform *schedulability analysis test*. Listed below are the edge conditions c_{od}:

- c_{11}: The previous job of the task τ_i has not exceeded its worst case execution time. In that case a new event v_1 is scheduled with the task's next job ($j_{i,k+1}$).
- c_{12}: There are no v_1 events on Λ scheduled for the current simulation time. This means that no new jobs instantiations are scheduled. Then a new v_2 event is programmed at the current instant in order to execute the highest priority job at the *ready queue*.
- c_{23}: The highest priority job $j_{i,k}$ could finalize before or at the scheduled time of the nearest v_1 event in Λ. Then it can complete its execution without interruptions, and a v_3 event (*Finalization*) is programmed.
- c_{24}: The highest priority job $j_{i,k}$ could not finalize before or at the scheduled time of the nearest v_1 event in Λ. Then it could possibly be preempted by a highest priority job, and a v_4 event (*Preemption*) is programmed.
- c_{32}: There are no v_1 events on Λ for the current simulation time and there is at least one job $j_{i,k}$ at the *ready queue*. Then a new v_2 event is programmed at the current instant in order to execute it.

The edge condition c_{32} avoids the duplication of a v_2 event in case of a v_1 event is scheduled at the same time. Notice that the edge conditions c_{23} and c_{24} are mutually exclusive.

F. Time delays

A v_d event should be scheduled for an instant $t_d \geq t$. In the *event graph* model this is expressed by associating a *time delay* $\delta_{od} \geq 0$ to each edge e_{od}, such that $t_d = t + \delta_{od}$.

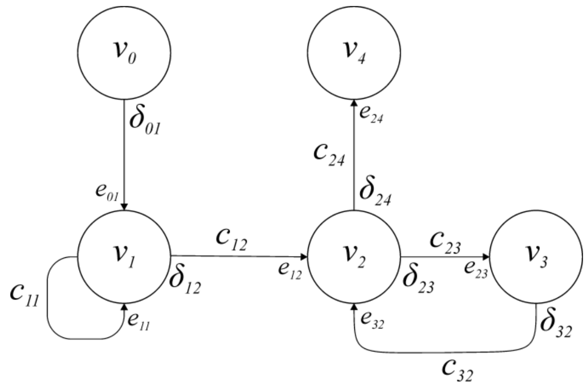
The edge e_{11} schedules a new event v_1, which represents a new job of a task i. The time delay δ_{11} is calculated with:

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

The event v_2 is scheduled at the current time by the edges e_{12} and e_{32}. Then $\delta_{12} = \delta_{32} = 0$. The time delay for the edge e_{23} is $\delta_{23} = C_i - c_{i,k}(t)$, which is the remnant execution time of the job $j_{i,k}$.

The edge e_{24} is traversed when the nearest v_1 event in Λ is scheduled for a time $t_1 < t + C_i - c_{i,k}(t)$. Then a v_4 event is scheduled at t_1 instant. So $\delta_{24} = t_1 - t$.

FIGURE I
FINAL EVENT GRAPH MODEL



G. Event execution

In this section we present the modifications to the system state (***S***) that each event performs when executes. These are the functions f_i for each v_i event.

The v_0 event (*Initialization*) schedules the task's initial jobs. So the first v_1 event of each task is programmed from v_0. The simulation clock is also initialized, generally with $t = 0$. Any other activity that should be done at *setup time* (i.e. worst case response time analysis) is performed at this event.

The v_1 event (*Arrival*) adds a new job $j_{i,k}$ to the scheduler *ready queue*. Then it schedules a new v_1 event for the next task job $j_{i,k+1}$ adding $(t + \delta_{11}, v_1(j_{i,k+1}))$ into Λ. Finally if the edge condition c_{12} is met, a v_2 event (*Execution*) is scheduled for that instant, $\Lambda \leftarrow (t, v_2)$.

In order to perform the *simulated execution* of the highest priority job at the *ready queue*, the method or routine that implements the *simulated scheduler* logic should be invoked at the v_2 event execution. If the edge condition c_{23} is satisfied, then a v_3 event (*Finalization*) is programmed $(\Lambda \leftarrow (t + \delta_{23}, v_3))$. Otherwise, a v_4 event (*Preemption*) is scheduled, $\Lambda \leftarrow (t + \delta_{24}, v_4)$.

The execution of a v_3 event (*Finalization*) should change ***S*** in order to indicate the highest priority job termination at *ready queue*. Then if the edge condition c_{32} is valid, a new v_2 event should be scheduled at the current simulated time in order to continue the execution of the other jobs that are at the *ready queue*. Similarly, the v_4 (*Preemption*) event invokes the necessary methods or routines to modify ***S*** in order to show the possible preemption of the current highest priority job.

Finally the v_5 event (*EndSimulation*) is scheduled at the instant t_{end} where the simulation should end. It must free any resources and invoke the auxiliary routines, like report generation.

H. Model execution example

An execution example of the model is shown at next; For this we will use the *RTS* $\Gamma(3) = \{(1, 3, 3), (1, 4, 4), (1, 6, 6)\}$. The next table shows the model evolution until $t = 6$.

TABLE II
MODEL EXECUTION EXAMPLE

t	Event	Λ	Executed job	Ready queue
0	v_0	$(0, v_1(j_{1,0})), (0, v_1(j_{2,0})), (0, v_1(j_{3,0}))$	-	-
0	$v_1(j_{1,0})$	$(0, v_1(j_{2,0})), (0, v_1(j_{3,0})), (3, v_1(j_{1,1}))$	-	$j_{1,0}$
0	$v_1(j_{2,0})$	$(0, v_1(j_{3,0})), (3, v_1(j_{1,1})), (4, v_1(j_{2,1}))$	-	$j_{1,0}, j_{2,0}$
0	$v_1(j_{3,0})$	$(0, v_2), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	-	$j_{1,0}, j_{2,0}, j_{3,0}$
0	v_2	$(1, v_3), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{1,0}$	$j_{2,0}, j_{3,0}$
1	v_3	$(1, v_2), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{1,0}$	$j_{2,0}, j_{3,0}$
1	v_2	$(2, v_3), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{2,0}$	$j_{3,0}$
2	v_3	$(2, v_2), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{2,0}$	$j_{3,0}$
2	v_2	$(3, v_3), (3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{3,0}$	-
3	v_3	$(3, v_1(j_{1,1})), (4, v_1(j_{2,1})), (6, v_1(j_{3,1}))$	$j_{3,0}$	-
3	$v_1(j_{1,1})$	$(3, v_2), (4, v_1(j_{2,1})), (6, v_1(j_{3,1})), (6, v_1(j_{1,2}))$	-	$j_{1,1}$
3	v_2	$(4, v_3), (4, v_1(j_{2,1})), (6, v_1(j_{3,1})), (6, v_1(j_{1,2}))$	$j_{1,1}$	-
4	v_3	$(4, v_1(j_{2,1})), (6, v_1(j_{3,1})), (6, v_1(j_{1,2}))$	$j_{1,1}$	-
4	$v_1(j_{2,1})$	$(4, v_2), (6, v_1(j_{3,1})), (6, v_1(j_{1,2})), (8, v_1(j_{2,2}))$	-	$j_{2,1}$
4	v_2	$(5, v_3), (6, v_1(j_{3,1})), (6, v_1(j_{1,2})), (8, v_1(j_{2,2}))$	$j_{2,1}$	-
5	v_3	$(6, v_1(j_{1,2})), (6, v_1(j_{3,1})), (8, v_1(j_{2,2}))$	$j_{2,1}$	-
6	$v_1(j_{1,2})$	$(6, v_1(j_{3,1})), (8, v_1(j_{2,2})), (9, v_1(j_{1,3}))$	-	$j_{1,2}$

At the instant $t = 0$ the *future event list* (Λ) contains the initials v_1 events, one for each task's first job. These events are generated by v_0. Then, the first v_1 event in Λ is executed, $v_1(j_{1,0})$. It schedules a new v_1 event at $t = 3$ for the next job of τ_i (which is $j_{1,1}$). This implies to put $(3, v_1(j_{1,3}))$ into Λ. The execution of the events $v_1(j_{2,0})$ and $v_1(j_{3,0})$ is analogous. When the last v_1 event on Λ is executed, the edge condition c_{12} is then valid. Therefore, a v_2 event is scheduled at $t = 0$. This event will simulate the execution of the highest priority job in the scheduler *ready queue* ($j_{1,0}$). As the job can finish before any future event at Λ, the edge condition c_{23} is satisfied. Then a v_3 event is schedule at $t = 1$. With no more events of any kind in Λ for $t = 0$, the simulation clock is then advanced to the instant time at which the next event is scheduled ($t = 1$). Then a new v_2 event is scheduled at $t = 1$ in order to continue the simulation of the jobs in the *ready queue*. The rest of the model execution presents an analogous behavior.

V. IMPLEMENTATION

Next a reference implementation in Java using the *SSJ* ([19]) simulation library is described. This library offers the package *simevents* for discrete event simulation, which has two main classes, *Simulator* and *Event*.

The class *Simulator* represents the simulation executive. It provides the simulation clock and the future event list Λ (offering multiple implementations). Also provides methods to start and stop the simulation.

The *Event* class represents an abstraction of an event. Each one of the events presented in section 4B should be implemented as a class which extends it (*Init*, *Arrival*, *Run*, *End* and *Preempt*). These classes must override the *actions*() method in order to perform the corresponding actions (that is the f_v functions listed in section 4G).

It is assumed that a collection or array with the *RTS* to simulate is provided. Also there must be auxiliary classes that implement the scheduler and other techniques to be evaluated.

The *Init* class (v_0 event) creates the initial instances of the *Arrival* class. These instances are scheduled at the appropriate times using the *schedule*(*delay*) method. The *Arrival* class adds its associated job into the *ready queue using the actions*() method,. If the condition c_{12} is met, it should schedule a new instance of *Run* (v_2) class at the current time. This class should invoke any method required to perform modifications at ***S***. If the edge conditions c_{23} or c_{24} are valid it should schedule an instance of *End* (v_3) or *Preempt* (v_4).

End class will invoke *Scheduler.finishTask*() method while *Preempt* will invoke *Scheduler.preemptTasks*(). This way the scheduler logic is decoupled from the event logic. If the edge condition c_{32} is met, *End* class must schedule a new *Run* class instance for the current time. The Simulator class from *SSJ* provides methods that help to verify the different edge conditions.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a discrete event model for the simulation of a *RTS*, based on the analysis of a *RTS* as *dynamic*, *discrete*, *non-linear* and *deterministic* system. The *event graph* technique was used as modeling tool due to its simplicity and ease of implementation. This model brings a framework in which simulation software can be developed using one of the many *DES* packages or libraries available. Also, this work serves as basis for future developments that extends the presented model; for example, for simulating *heterogeneous RTS*.

VII. REFERENCES

[1] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "STRESS: A simulator for hard real-time systems," *Software: Practice and Experience,* 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.