Una biblioteca de clases en Java para Simulación y apoyo al estudio de Sistemas de Control

Miguel Ángel Redondo Duque
Departamento de Informática - Universidad de Jaén
E.U.P. de Linares
C/ Alfonso X el Sabio, 28. 23700 - Linares
Teléfono 953 649585
E-mail: mredondo@ujaen.es

Resumen

Este trabajo presenta una herramienta para el estudio mediante simulación, de procesos industriales y sistemas de tiempo real, junto con la metodología que se ha seguido para su desarrollo. Contrariamente a lo que otras herramientas ofrecen, no se trata de un entorno cerrado de simulación, si no que es una librería de clases para Java que permite simular el comportamiento de algunos elementos físicos elementales, para que partiendo de estos, mediante las directrices de diseño que marca la orientación a objetos, se puedan construir otros más complejos, siguiendo la misma metodología que se ha empleado aquí. Con estas herramientas los alumnos que estudian, con detalle, teorías de control digital de procesos, podrán realizar prácticas avanzadas, llegando hasta el mismo núcleo del problema en orden a analizar los requisitos no-funcionales del sistema, modificando el enfoque actual en el que se lleva a cabo el análisis de requisitos de seguridad (dependability) y restricciones temporales de estos sistemas después de desarrollar completamente un prototipo del sistema.

1. Introducción.

La gran complejidad de los sistemas de control modernos hacen muy difícil especificar las propiedades de dichos sistemas y por lo tanto verificarlas. Las descripciones de propiedades utilizando *lenguaje natural* deberían evitarse y emplear notaciones de especificación basadas en métodos formales, no obstante, aunque en la actualidad están proliferando estas técnicas, sigue siendo difícil recoger todas las propiedades de los sistemas en una primera aproximación, ya que es en los primeros prototipos cuando empiezan a quedar claros los requisitos del sistema. Es ampliamente conocido que los requisitos de los sistemas de tiempo real son mucho más estrictos que los de otro tipo de sistemas [Stankovic88]. La descripción de estos sistemas no sólo se refiere a las propiedades que se derivan de su funcionamiento, sino que deben contemplar también requisitos no-funcionales para garantizar que se cumplen las propiedades específicas de tiempo real, tales como: *oportunidad* (*timeliness*) y *predecibilidad*. La primera se refiere al hecho de que los sistemas de tiempo real han de estar siempre disponibles para reaccionar con rapidez a estímulos del entorno. La segunda, se refiere a que la respuesta ha de producirse dentro de unos límites de tiempo pre-establecidos. Estos sistemas han de satisfacer las restricciones temporales impuestas por el entorno. La pérdida de un *tiempo límite* (*deadline*) no es admisible en sistema de tiempo real duros.

Por otro lado, el fallo de muchos sistemas de tiempo real puede tener como resultado desastres con riesgo de vidas, que atentan a la ecología y/o representan pérdidas económicas importantes. Esta idea indica la importancia de los métodos y técnicas que ayudan a desarrollar herramientas de análisis que aseguren que se satisfacen las propiedades de *seguridad* (*dependability*).

La complejidad manifiesta de este tipo de sistemas se debe plasmar en el estudio de los mismos, pero de una forma constructiva, es decir, es importante disponer de herramientas que faciliten el estudio de estos sistemas mediante simulación de su comportamiento, con el máximo nivel detalle posible, de forma que dicha simulación verdaderamente esté cargada de realismo, permitiendo obtener algoritmos de control de calidad. Por otro lado, resulta didácticamente interesante, que un alumno que estudia técnicas de control pueda extraer conclusiones manejando una visión (la simulación), próxima a la realidad, de lo que constituyen este tipo de sistemas.

1.1 Técnicas de simulación de sistemas.

Usar modelos o marcos de simulación para el desarrollo y validación de sistemas de control ofrece una gran ventaja respecto a usar dispositivos físicos reales. Se abaratan los costes si antes de construir el modelo físico real se ha utilizado un prototipo para validar los requisitos. Además, usar máquinas reales puede ser peligroso por la posibilidad de dañar algún dispositivo debido a un error de operación, y desde luego su contenido didáctico puede resultar completamente nulo.

Para modelar y simular sistemas industriales se utilizan los conceptos de tiempo continuo y eventos discretos. Para simplificar los modelos la mayoría de las acciones que implican tiempo continuo se abstraen y pasan a convertirse en eventos discretos, por ejemplo, el control de la posición de los brazos de un robot puede restringirse sólo a un conjunto finito de posiciones.

El trabajo en entornos de simulación se puede clasificar en tres categorías:

a) Lenguajes de simulación
b) Simuladores concretos
c) Marcos de simulación

Los primeros son lenguajes de programación con características especiales que hacen más fácil escribir o programar simuladores [Kiviat93]. Estos lenguajes tienen el poder de expresión de los lenguajes de uso general y gran flexibilidad para el diseño e implementación de simuladores concretos. El inconveniente es el gran esfuerzo que se necesita para programar el simulador. Ejemplos de lenguajes de simulación y de bibliotecas de funciones se pueden encontrar en [Scwetman94].

Los segundos, son aplicaciones concretas que se crean para simular el comportamiento de un determinado sistema. En éste caso, si la herramienta es fácil de utilizar, la simulación del sistema puede ser realizada en poco tiempo. Un ejemplo de este tipo es el utilizado en PERTS [Liu93], y la simulación conocida como *Production Cell* [Lewerentz95].

Los terceros proporcionan un marco y una implementación o herramienta de simulación que permite al que la utiliza, definir y crear nuevas simulaciones. Los marcos de simulación son, al igual que los simuladores concretos, fáciles de utilizar y fácilmente extensible a otros dominios. Este enfoque se utiliza en NEST [Ymini90]. El enfoque basado en marcos de simulación ofrece la posibilidad de extensibilidad y

reusabilidad de componentes. El programador tiene la posibilidad de programar nuevos componentes, utilizando para ello algunos que ya están programados o partiendo de cero, ya que el simulador está construido en un marco de simulación perfectamente definido y documentado. Al igual que los segundos, los marcos de simulación se aplican a una determinada clase de sistemas, pero pueden ser ampliados fácilmente para incluir nuevas clases de sistemas.

Dentro de este tipo de sistemas, cabe destacar el trabajo actualmente desarrollado en la Universidad de Twente, donde cabe destacar al grupo liderado por Gerald Hilderink que integra a investigadores de dos instituciones de dicha universidad: *Control Laboratory* y *Faculty of Technical Sciences*. Este grupo ha trabajado en la automatización de herramientas de diseño con la suficiente difusión y aceptación por parte de la industria, como resultado de su trabajo se han desarrollado varias herramientas: 20-sim [Broenink90], THESIS [Wijbrans93] y S&A [Sunter94].

20-sim es una herramienta para la modelización y simulación del comportamiento de sistemas físicos que permite generar código para la parte de control del sistema, a partir de una especificación del mismo. THESIS (Twente Hierarchical Embedded Systems Implementation by Simulation) es una herramienta para la especificación y validación de algoritmos de control que permite conseguir una correspondencia entre el modelo físico y el sistema de control en un computador paralelo. S&A es una herramienta que se utiliza para la asignación y planificación automática de procesos paralelos en una aplicación de tiempo real.

Todas las herramientas anteriores se están integrando en un entorno de desarrollo cuyo objetivo es servir de soporte durante toda la fase de diseño del ciclo de vida de SCETRs (Sistemas de Control Empotrados de Tiempo Real).

Este trabajo utiliza metodologías para el desarrollo de simuladores como [Capel94], [Burns95] o [Rumbaugh91] y sobretodo lenguajes de desarrollo orientados a objetos como Java. De esta forma se conseguirá desarrollar, además de un simulador concreto, técnicas metodológicas que permitan incorporar al simulador nuevas funcionalidades, según las necesidades de sistemas concretos. Un trabajo previo interesante sobre construcción de simuladores utilizando POO y Java es [Buss96].

En este artículo se lleva a cabo una descripción del modelo matemático empleado para la representación de los elementos y procesos físicos, apuntando algunas pinceladas sobre la metodología de diseño e implementación que se lleva a cabo.

2.Técnica empleada.

2.1. Modelo matemático.

2.1.1. Sistemas continuos.

Muchos sistemas dinámicos, ya sean mecánicos, térmicos, hidráulicos, etc., pueden ser caracterizados por ecuaciones diferenciales o sistemas de ecuaciones diferenciales. Se puede obtener la respuesta de un sistema dinámico a una entrada, o función excitadora, resolviendo dichas ecuaciones diferenciales. Para obtener las ecuaciones se usan las leyes físicas que gobiernan el sistema en particular, por ejemplo, las leyes de Newton para sistemas mecánicos.

La descripción matemática de las características dinámicas de un sistema se denomina *modelo matemático*. Para el análisis o simulación de un sistema dinámico, el primer paso a realizar es determinar su modelo. Esta es la parte más importante, ya que de ella depende el éxito del análisis.

Un sistema no tiene un modelo matemático único, sino que los modelos pueden tomar distintas formas según sea el sistema particular de que se trate y en qué circunstancias se encuentre. De todas las formas posibles que tomará el modelo, unas serán mejores o más adecuadas que otras, y según se tome una u otra los cálculos posteriores se simplificarán o se complicarán.

Por otra parte, para el análisis de propiedades dinámicas de sistemas con entrada o salida única, la representación de la *función de transferencia* o *transmitancia* se puede considerar como la más indicada.

Para la obtención de la función de transferencia se llevan a cabo una serie de pasos:

- Planteamiento de la ecuación diferencial del sistema: $f(t)$
- Aplicar la transformada de Laplace de la ecuación anterior, suponiendo todas las condiciones iniciales nulas: $F(s) = \lambda[f(t)]$
- Obtener la función de transferencia como la relación entre la salida *Y(s)* respecto a la entrada *X(s)*: $G(s) = \dfrac{Y(s)}{X(s)}$

2.1.2 Sistemas discretos. Transformada Z.

Hay que tener presente que el objetivo que se persigue es la simulación de procesos, generalmente de naturaleza continua, mediante algoritmos cuya naturaleza es inherentemente discreta. Por lo tanto, será necesario recurrir a herramientas matemáticas que permitan trabajar con sistemas discretos. La transformada Z es la principal herramienta analítica para tal fin.

La transformada Z [Leigh85] [Phillips87] es a los sistemas discretos lo que la transformada de Laplace es a los sistemas continuos, donde el símbolo *z* es análogo al símbolo *s*, conceptualmente *z* se puede asociar con un desplazamiento de tiempo en una ecuación de diferencias[1], del mismo modo que *s* se puede asociar con una diferenciación en una ecuación diferencial.

A una función discreta $f^*(t)$, equivalente a una señal o función muestreada (señal continua multiplicada por un tren de impulsos), se le puede calcular su transformada de Laplace, de la siguiente forma:

$$F^*(s) = \lambda\{f^*(t)\} = \int_0^\infty f^*(t)e^{-st}dt$$

La función $f^*(t)$ solo toma valores distintos de cero en aquellos instantes de tiempo donde $t = kT$, tomando *k* valores enteros

$$F^*(s) = \int_0^\infty f^*(t)e^{-st}dt = \sum_{k=0}^{\infty} f(kT)e^{-skT}$$

[1] Se trata de una forma particular de representar los sistemas discretos lineales.

La transformada Z de $f(t)$ debe ser igual a la transformada de Laplace de $f^*(t)$, es decir

$$Z\{f(t)\} = \lambda\{f^*(t)\}$$
$$F(z) = F^*(s) = \sum_{k=0}^{\infty} f(kT)\left(e^{st}\right)^{-k}$$

Finalmente, haciendo $z = e^{sT}$, se obtiene

$$F(z) = \sum_{k=0}^{\infty} f(kT) z^{-k}$$

Luego se puede obtener $F^*(s)$ a partir de $F(z)$ y viceversa con solo hacer el cambio de variable correspondiente.

No obstante, y para facilitar su cálculo se puede recurrir a la utilización de la *teoría de residuos*, de tal forma que

$$F(z) = Z\{F(s)\} = \sum_{polos \in F(s)} \text{Residuos}\left[F(s)\frac{1}{1-e^{st}z^{-1}}\right]$$

teniendo presente que

$$\text{Re}\,siduo(z = z_0) = \frac{1}{(m-1)!}\lim_{z \to z_0}\frac{d^{m-1}}{dz^{m-1}}\left[(z-z_0)^m f(z)\right]$$

donde *m* es el orden del polo en cuestión.

2.1.3.Transformada Z inversa.

De la misma forma que hacemos uso de la transformada Z, necesitaremos utilizar su transformada inversa, para retornar al espacio del tiempo (discreto en nuestro caso). La expresión que calcula Z^{-1} es

$$f_k = \frac{1}{2pj}\oint_C f(z) z^{k-1} dz$$

donde *C* es un camino cerrado que recoge todas las singularidades del integrando. Nuevamente esta expresión resulta mucho mas fácil de evaluar utilizando la teoría de residuos.

Una vez que aplicamos la transformada Z inversa y volvemos al espacio del tiempo, la expresión que obtenemos es fácilmente implementable mediante un algoritmo, como se verá en el ejemplo que se muestra a continuación.

2.1.4. Ejemplo.

Considérese un elemento donde el comportamiento de su salida en función de su entrada se rige por la siguiente expresión (en el espacio de Laplace),

$$F(s) = \frac{1}{s+1}$$

se pretende obtener un algoritmo discreto que simule el comportamiento de dicho elemento.

F(s) tiene un único polo en s=-1, luego

$$F(z) = \mathrm{Z}\{F(s)\} = \sum_{s=-1} \mathrm{Re}siduos\left[F(s)\frac{1}{1-e^{st}z^{-1}} \right]$$

$$F(z) = \mathrm{Re}siduo(s=-1) = \lim_{s\to -1}(s+1)\frac{1}{s+1}\frac{1}{1-e^{sT}z^{-1}} = \frac{1}{1-e^{-T}z^{-1}}$$

Operando

$$Y(z)\left[1-e^{-T}z^{-1}\right] = X(z)$$

$$Y(z) - Y(z)e^{-T}z^{-1} = X(z)$$

Calculando la transformada Z^{-1} obtenemos

$$y(k) - e^{-T}y(k-1) = x(k)$$

$$y(k) = x(k) + e^{-T}y(k-1)$$

Ahora fácilmente se puede convertir en un algoritmo, quedando de la siguiente forma:

```
INICIO Simulación
    salida ← 0
    MIENTRAS verdad HACER
        LEER(entrada)
        salida ← entrada + EXP(-T) * salida
    FIN-MIENTRAS
FIN Simulación
```

2.1.5. Proceso general.

Resumiendo, para obtener el algoritmo que simula el comportamiento discreto de cualquier elemento o proceso físico continuo (como caso general) seguiremos los siguientes pasos:

- Planteamiento de la ecuación diferencial del elemento: $f(t)$
- Aplicar la transformada de Laplace de la ecuación anterior, suponiendo todas las condiciones iniciales nulas: $F(s) = \lambda\left[f(t) \right]$

- Obtener la función de transferencia como la relación entre la salida *Y(s)* respecto a la entrada *X(s)*: $F(s) = \dfrac{Y(s)}{X(s)}$
- Considerar un determinado periodo de muestreo, *T*, y por tanto, discretizar la función de transferencia, obteniendo $F^*(s)$
- Aplicar la transformada Z a $F^*(s)$ obteniendo $F(z)$.
- Calcular la transformada Z inversa y obtener la ecuación en diferencias, y de ésta última deducir el algoritmo que simula el comportamiento del elemento.

2.1.6. Elementos de retención.

Es necesario determinar los efectos que produce el muestreo de una señal en tiempo continuo. Generalmente este muestreo es llevado a cabo por elementos conversores Analógico/Digital y Digital/Analógico, denominados en teoría de muestreo y discretización como mantenedores, y por la aproximación matemática que realiza para su modelado se les añade la coletilla "*de orden cero*" (*Zero Order Hold*). Estos elementos tienen incidencia sobre el sistema total, de tal forma que la ecuación en el espacio *z* que se obtiene a partir de $F(s)$ queda definida como:

$$\overline{F_{h_0} F(z)} = (1 - z^{-1}) \mathrm{Z}\left[\frac{F(s)}{s}\right]$$

2.1.7. Otros métodos de discretización.

Existen otras posibilidades que pueden ser útiles a la hora de realizar la aproximación de ecuaciones diferenciales, estas posibilidades pasan por aproximar las integrales y derivadas de la función de transferencia en el espacio *s* mediante técnicas de aproximación rectangular hacia delante y hacia atrás, e integración trapezoidal.

$$F_1(z) = F(s)\Big|_{s=(z-1)/T} \qquad \textit{hacia delante}$$

$$F_1(z) = F(s)\Big|_{s=(1-z^{-1})/T} \qquad \textit{hacia atrás}$$

$$F_1(z) = F(s)\Big|_{s=\frac{2}{T}\frac{(z-1)}{(z+1)}} \qquad \textit{trapezoidal}$$

No obstante estas técnicas suponen optimización en tiempo, pero no siempre son aplicables, su bondad depende muy directamente del periodo de muestreo (*T*) que se utilice en cada caso, por lo tanto, deben considerarse como métodos muy específicos y poco generales.

2.1.8. Redes neuronales.

Es necesario tener en cuenta que aunque el mundo que nos rodea pueda ser modelado mediante ecuaciones diferenciales, en muchos casos encontrar la función de transferencia que modela el comportamiento del elemento que estudiamos puede ser todo un problema, y si no encontramos dicha función de transferencia todo el desarrollo realizado en los apartados anteriores carece de sentido. En estos casos puede ser muy

oportuno recurrir al empleo de redes neuronales para encontrar los coeficientes y por tanto el algoritmo que simula el comportamiento del elemento en cuestión. De todas formas, esta posibilidad hay que considerarla como un método muy específico y nunca considerarla como la panacea, debido al costo de entrenamiento que acarrea y posterior aplicación a elementos muy concretos.

2.1.9. Sistemas causales. Problemas derivados.

Un sistema *G* es causal, si y solo si, *g(k)=0* para todo *k* menor que cero, dicho de otra forma, un sistema es causal cuando la respuesta nunca precede a la excitación o entrada, lo que nos viene a decir que no se pueden utilizar muestras futuras de la entrada para determinar la salida en el momento actual.

Atendiendo a la función de transferencia en el dominio *z* (dominio de la transformada *Z*), para que un sistema sea causal dicha función de transferencia debe contar con un número de polos (raíces del denominador de la función) mayor o igual al número de ceros (raíces del numerador de la expresión). En caso de no cumplirse esta condición, hay que recurrir a otro tipo de aproximaciones en la discretización (véase apartado 0), ya que, la aplicación del método general da como resultado un algoritmo para el cálculo de la salida en un instante k_0, pero en función de valores de entrada en instantes k, tales que $k>k_0$.

Generalmente, este tipo de problemas surgirán cuando la función de transferencia de un sistema cuente con mayor número de operaciones de integración que de operaciones de derivación, ya que, el problema precisamente viene de las operaciones de integración.

2.2. Diseño e Implementación.

En este caso, como en otros muchos, la implementación condiciona el diseño, se pretende llevar a cabo una implementación de una biblioteca de clases, obviamente para un lenguaje de programación orientado a objetos: JAVA. Aunque en este artículo no se presenta el diseño completo si que se apuntan las directrices que se siguen, en posteriores avances de este trabajo se llevará a cabo un diseño completo, utilizando como herramienta de especificación UML "*Lenguaje Unificado para Modelado*", pensando en un enfoque distribuido, utilizando el concepto de "*Communicating Sequential Processes*" (CSP) [Hoare78] como mecanismo de sincronización, utilizando la librería de clases desarrollada en [Hiderink97] que enriquece el lenguaje Java, añadiéndole las posibilidades de sincronización estilo CSP. Con la utilización de Java, en principio,

se dispondrá de todo el paralelismo que sea capaz de proporcionar la máquina en la que se lleve a cabo la ejecución junto con las posibilidades que proporcione el intérprete (o incluso compilador) de Java con el que se cuente.

Con el diseño e implementación que anteriormente se ha apuntado, la construcción de un sistema para su simulación consistirá en tomar una serie de clases, cada una que implemente el comportamiento de los componentes elementales del sistema, como si se tratase de elementos estructurales de un entorno integrado de simulación, pero desde el punto de vista del programador de algoritmos de control, para posteriormente establecer los canales de comunicación para que dichos elementos puedan interactuar entre sí, dando lugar al comportamiento global del sistema.

3. Conclusiones.

El trabajo desarrollado pretende encontrar un método numérico que fuese suficientemente flexible para soportar, de un modo general, la simulación de cualquier proceso físico expresado mediante su función de transferencia.

Nos encontramos con ciertas restricciones: derivadas de que el sistema que se obtiene puede no ser causal, y por lo tanto implementable en un computador digital mediante un algoritmo. En estos casos hay que recurrir a otros métodos de discretización consistentes llevar a cabo aproximaciones de las ecuaciones diferenciales, no obstante estas aproximaciones también nos introducen ciertas limitaciones para que se puedan aplicar con cierta precisión (relación entre la frecuencia de muestreo y la frecuencia de trabajo del sistema).

En general, podemos decir que cuando en las funciones de transferencia no aparecen integrales es altamente probable obtener buenos resultados aplicando el general, sin embargo, cuando aparecen integrales habrá que recurrir a realizar aproximaciones de la función de transferencia (apartado 0), siempre que las condiciones de trabajo del sistema lo permita. No obstante, se puede extraer una metodología de trabajo que permite construir multitud de clases que modelen el comportamiento de otros tantos sistemas físicos, y a partir de esas clases elementales, siempre a nivel de programación, con la sencillez que brinda la orientación a objetos junto con un modelo que permite un enfoque distribuido como es el CSP, construir sistemas verdaderamente completos y complejos.

Hay que destacar que este desarrollo aprovecha las ventajas que ofrece un lenguaje como Java, es decir, orientación a objetos y perfectamente integrado en el Web, con todo lo que esto conlleva: simulación a distancia, trabajo colaborativo, intercomunicación profesor-alumno, etc., conceptos que ya van sonando más a presente que a futuro.

Todavía queda trabajo por realizar hasta llegar a tener una completa librería de clases que recojan el comportamiento de gran número de los componentes elementales que nos podemos encontrar en los procesos físicos, realizando un estudio de los mismos para optar por el algoritmo de simulación que mejor refleje su comportamiento.

4. Referencias.

[Broenink90] Broenink, J.F. *Computer Aided Physical Modelling and Simulation: a Bond Graph*

Approach. PhD Thesis, U. Twente, 1990.

[Burns95] Burns, A., Wellings, A. *HRT-HOOD: A Structured Design Method for Hard Real*

Time Ada Systems. Elsevier, 1995.

[Buss96] Buss, A.H., Stork, K.A. *Discrete Events Simulation on The World Wide Web Using*

Java. Proceedings of the Winter Simulation Conference. De: J.M. Charnes, D.J. Morrice, D.T.

Brummer and J.J. Swain. Colonado, California, 1996.

[Capel94a] Capel, M., Troya, J.M., A. Palma. *A Methodological Scheme and Tool for Program*

Transformation with Transputer Sustems. En Joubert, G.R., Trystam, D., Peters, F.J., Evans,
D.J. 'Parallel Computing: Trends and Applications'. Elsevier Science B.V., Amsterdam, 1994.

[Capel94b] Capel, M., Troya, J.M., *An Object Based tool and Methodological Approach for*
Distributed Programming. Software Concepts and Tools, 15, pp.177-199, 1994.

[Hiderink97] Hiderink, G., Broenik J., Vervoot W.,Bakkers A. "*Communicating Java Threads*".
University of Twente, dpt. EE., Control Laboratory. 1997.

[Hoare78] Hoare, C.A.R., *Communicating sequential processes.* ACM 21, 8, Agosto 1978.

[Kiviat93] Kiviat, P. *A Brief Introduction to Discrete-Event Simulation Programming*
Languajes. ACM SIGPLAN Notes, 28 (3), March 1993. Proceedings of History of Programming Languajes Conference (HOPL-II).

[Leigh85] Leigh, J.R. "*Applied Digital Control Theory. Design and Implementation*". Prentice-
Hall 1985.

[Lewerentz95] Lewerentz, C. and Lindner, T. (eds.) *Formal Development of Reactive Systems:*
Case Study Production Cell. LNCS 981, Springer-verlag, 1995.

[Liu93] Liu, J., Redondo, J.L., Deng, Z., Tia, T.S., Bettati, R., Silberman, A. Storch, M., Ha,
R. y Shih, W. *PERTS: A Prototyping Environment for Real-Time Systems.* In Proceedings of the
14th IEEE Real-Time Systems Symposium, pp. 184-188, Raleigh-Durham, North Carolina,
1993.

[Phillips87] Phillips, C.L., Nagle, H.T. "*Sistemas de Control Digital. Análisis y Diseño*".
Colección Ciencia Electrónica. Ediciones G.Gili, 1987.

[Rumbaugh91] Rumbaugh et al. *Object Oriented Modeling and Design.* Prentice-Hall, 1991.

[Scwetman94] Schwetman, H. *Csim 17: A Simulation Model-Building Toolkit.* En Winter
Simulation Conference Proceedings, pp. 464-470, New York, NY, 1994. ACM Press.

[Stankovic88] Stankovic, J.A. *Misconceptions About Real-Time Systems -A Serious Problem for the*
Next Generation Systems. pp. 10-19. IEEE Computer Magazine, 1998.

[Sunter94] Sunter. *J.P.E. Allocation, Scheduling & Interfacing in RT Paralel Control Systems.*
PhD Thesis, U.Twente, 1994.

[Wijbrans93] Wijbrans, K.C.J. *Twente Hierarchical Embedded Systems Implementation by*
Simulation. PhD Thesis. U.Twente, 1993.

[Ymini90] Ymmini, Y., Bacon, D. *Nest: A Network Simulation and Prototyping Testbed.*
Communications of the ACM, 33 (10), pp. 63-74, 1990.