Revista Iberoamericana de Automática e Informática Industrial
CEA IFAC comité español de automática
RIAI
ISSN: 1697-7912. Vol. 2, Núm. 4, Octubre 2005, pp. 66-77
http://riai.isa.upv.es

LIBRERÍA JAVA PARA LA SIMULACIÓN DE SISTEMAS DE EVENTOS DISCRETOS Y SU APLICACIÓN EN LA GERENCIA HOSPITALARIA

R.M. Aguilar, C.A. Martín, I. Castilla, V. Muñoz, L. Moreno

Dpto. Física F. E. Electrónica y Sistemas.
Universidad de La Laguna
Avda. Astrofísico Francisco Sánchez s/n
38271 La Laguna, España
e-mail: raguilar@ull.es

Resumen: A pesar de las herramientas de programación que ofrecen los entornos de simulación de eventos discretos que existen en el mercado, éstas no siempre facilitan la codificación eficiente de la dinámica de interés en ciertos campos de aplicación. En este artículo se describe una librería, implementada en Java, que dispone de un conjunto de funciones para facilitar al ingeniero el modelado y simulación de un sistema de eventos discretos, y en concreto de un centro hospitalario, abstrayéndose de los detalles técnicos de bajo nivel comunes en todas las implementaciones de este tipo. *Copyright © 2005 CEA-IFAC*

Palabras Clave: simulación, sistema de eventos discretos, programación orientada a objetos (Java), modelado orientado al proceso.

1. INTRODUCCIÓN

En los hospitales, como en toda organización de servicios, la toma de decisiones debería estar soportada por una herramienta que permitiera a la gerencia predecir los efectos de la aplicación de distintas acciones de control para aumentar su eficiencia. Sin embargo, el uso de éstas no es práctica común en los hospitales y se presentan situaciones en las que la gerencia hospitalaria estudia a posteriori los efectos (tanto positivos como negativos) de las acciones ya aplicadas. El problema aumenta debido a que este estudio se hace en función de una gran cantidad de datos que son tratados únicamente con hojas de cálculo.

Las técnicas de simulación han demostrado ser muy útiles como herramienta de análisis para minimizar el riesgo en la toma de decisiones, ya que permiten reducir la incertidumbre acerca de cómo afectarán los diferentes cambios al sistema existente. En este trabajo se describe las principales características de un entorno de simulación que ha sido desarrollado para predecir los efectos que tendrán en el hospital (debido a las interrelaciones existentes entre los departamentos) las acciones de control realizadas. Esta herramienta permitirá experimentar sobre las distintas redistribuciones de recursos, facilitando la selección de las decisiones que cumplan los requerimientos de costes impuestos.

Un hospital es un sistema formado por un gran número de partes (pacientes, recursos humanos y recursos materiales) y muchas interrelaciones entre estas partes (el paciente para ser atendido necesita de la cooperación de una variedad de recursos). El método tradicional de encontrar una solución analítica a partir de un modelo matemático del sistema es complejo, aunque el modelado de determinados departamentos del hospital ha sido realizado con éxito cuando éstos han sido descritos como sistemas de colas (González, *et al.*, 1994;

Barber 1993). Sin embargo, cuando se trata de modelar todo el hospital incorporando todas las interrelaciones entre los diferentes servicios, así como la compatibilidad con los sistemas de información existentes, se requiere disponer de un modelo donde se contemple el flujo de cada paciente.

Un Sistema de Eventos Discretos se caracteriza porque las propiedades de interés del sistema cambian únicamente a intervalos no regulares de tiempo. La secuencia de instantes en los cuales el estado del sistema puede cambiar, obedece a un patrón aleatorio (Guasch, *et al.*, 2002). El flujo de pacientes por los distintos departamentos del hospital puede verse como un sistema de eventos discretos (Fishwick 1995; Banks, *et al.*, 2001). Para el modelado de este sistema se utilizó inicialmente Redes de Petri (RdP). El resultado fue un modelo demasiado complejo para ser utilizado por la gerencia: en el estudio de distintas distribuciones de recursos el gerente debía cambiar la RdP ya que habían recursos modelados por transiciones. La siguiente opción analizada fue utilizar el lenguaje de simulación MedModel (MedModel, 2005), del grupo de programas ProModel, diseñado para la simulación de hospitales. Esta herramienta resultó ser muy autocontenida y con poca flexibilidad para definir procesos no estándares:

- Sólo se puede realizar un grupo reducido de acciones por parte de los objetos.
- Obliga al modelado de detalles (tiempo que tarda el paciente de llegar de la sala de espera al consultorio).
- Sólo permite conocer estadísticos prefijados (no es posible saber la longitud de una cola).
- No permite almacenar el estado del sistema en un instante dado.
- Ninguna facilidad para comunicarse con otras aplicaciones

Finalmente se desarrolló el modelo de la evolución de los pacientes por los distintos departamentos del hospital en el lenguaje de simulación ModSim II (CACI 1995). En este caso sólo se pudo modelar hospitales prototipo y los excesivos tiempos de ejecución de las simulaciones impidieron escalar estos modelos (Moreno, *et al.*, 2000).

En este artículo, se presenta el desarrollo de una librería de simulación de sistemas de eventos discretos, basada en el paradigma de "modelado orientado al proceso", usando Java como lenguaje de programación (Magee, Kramer, 1999). Esta herramienta se ha diseñado de manera que el ingeniero pueda centrarse únicamente en el modelado del problema, quedando a cargo de la librería la sincronización de los eventos y los detalles de la gestión de la programación.

La organización de este artículo es la siguiente: en el segundo apartado se introduce los conceptos relacionados con el modelado orientado al proceso y su aplicación en sistemas del tipo de un centro hospitalario. En el tercer apartado se hace una breve descripción de la librería, desarrollándose en los dos apartados siguientes las dos formas de hacer evolucionar el tiempo de simulación. El apartado 6 muestra una serie de ejemplos ilustrando algunas de las funcionalidades de la librería.

2. EL MODELADO ORIENTADO AL PROCESO

Un proceso es una secuencia ordenada de eventos interrelacionados separados por el paso del tiempo. Esta secuencia describe el paso de un elemento (individuo, información, ...) a través del sistema. Para modelar un sistema utilizando una metodología basada en el evento (RdP), el analista debe dividir el proceso (ciclo de vida) de cada elemento en las partes más elementales, determinando todas las posibles consecuencias de cada evento. Sin embargo en una metodología basada en el proceso, se elimina esta atomización, y se define el proceso completo de cada elemento como un único bloque lógico (Pidd 1998). Para utilizar una aproximación orientada al proceso hay que definir la secuencia de pasos (eventos o serie de eventos) para cada transacción que ocurre en el sistema. Los objetos requeridos por una transacción durante su movimiento a través del sistema son definidos como recursos.

La simulación orientada al proceso o basada en el proceso se usa mucho en la simulación de sistemas que se encuentra en la naturaleza (ej. colmena). Cada parte de estos sistemas tiene un conjunto de reglas que gobiernan el comportamiento de cada individuo durante su ciclo de vida (nacimiento, limpieza, alimentación, ...). Cuando se representan estas reglas en el ordenador se está creando una instanciación artificial de un componente. Cada uno de estos procesos hace uso de recursos. A menudo, los modelos de simulación deben formalizar la competición de los elementos que integran el sistema para poder acceder a los recursos. Los procesos tendrán que esperar en cola antes de utilizar los recursos (ej. paquetes esperando por el uso de una línea de red). Los recursos no tienen que corresponderse necesariamente con entidades físicas del sistema. Para el proceso de una abeja se define como recurso el conocimiento acerca de la localización de miel, el cual sólo puede ser obtenido cuando otra abeja hace que el recurso esté disponible "con su baile" (Langton, *et al.*, 1995; Gilbert, *et al.*, 1993; Barceló, *et al.*, 1999a,b,c).

La simulación orientada al proceso es apropiada cuando se puede reducir el sistema a un conjunto de procesos que adquieren y abandonan recursos sin pensar mucho en el entorno en el cual la entidad

existe. Esto es especialmente indicado en simulaciones de sistemas de redes, procesos de manufactura o colas en distintas oficinas. Sin embargo, cuando el sistema no tiene un claro intercambio de recursos entre sus procesos (ej. comunicación de abejas con otras) y existe un profundo sentido del entorno en el cual el proceso vive (ej. colmena de abejas) es difícil construir un modelo que es esencialmente un conjunto de colas. En estos casos para realizar simulaciones se debe incrementar el poder de comunicación a los procesos y definirles claramente el entorno donde viven, por lo que se hace necesaria la utilización de agentes (Langton, *et al.*, 1995).

En el caso del hospital tenemos un sistema donde el conjunto de pacientes representan la colectividad de procesos. Estos procesos se comunican mediante la competición por recursos (materiales o humanos), lo que hace que sea un sistema de eventos discretos adecuado para ser modelado mediante una aproximación orientada al proceso.

El cuello de botella en una simulación de este tipo está en la gestión de la lista de eventos pendientes, que es la estructura de datos en la que se registran todos los elementos activos (en el caso del hospital cada paciente y cada uno de los recursos), y sus actividades.

El algoritmo de gestión de la lista de eventos pendientes utiliza una estructura de datos como la de la figura 1. En ella deben aparecer todos los elementos que intervienen en el sistema y que generan acciones, y se ordenan atendiendo al tiempo en que tienen que ser lanzadas sus actividades. Así el elemento con la actividad a ser ejecutada más próximamente será el primero de la lista. Cuando la primera actividad se realiza, esta estructura de datos debe ser de nuevo ordenada de forma que siempre quede en la cabeza de la lista el elemento con la actividad más próxima a ejecutar. La gestión de esta lista hace que la ejecución sea cada vez más lenta, a medida que el número de elementos aumenta.

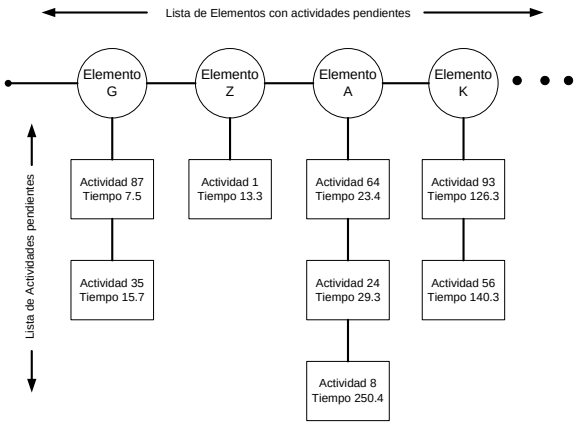

Figura 1. Lista de eventos pendientes.

3. LIBRERÍA JAVA PARA LA SIMULACIÓN DE SISTEMAS DE EVENTOS DISCRETOS

La librería desarrollada proporciona al programador de aplicaciones de simulación las clases básicas (con sus atributos y métodos) para que pueda importarlas o en su caso extenderlas, y facilitar en cualquier modo el modelado y simulación de un sistema de eventos discretos. Esto es, la utilidad de la librería sería posibilitar a cualquier desarrollador definir el sistema en estudio utilizando las clases de objetos implementadas y sobrescribiendo aquellos métodos que fueran necesarios para representar las características particulares de este sistema. Todo ello, abstrayéndose de la engorrosa tarea de programar cada uno de los procedimientos utilizados de forma genérica en una simulación de eventos discretos (SIGHOS, 2005).

Como se indicó anteriormente, la metodología de modelado usada es la orientada al proceso. Esto es, el sistema se caracteriza porque los elementos van discurriendo por etapas que pueden ser unas u otras en función del estado del propio elemento. Dentro de cada etapa estos elementos realizan diferentes actividades. Para poder efectuarlas hacen uso de recursos pertenecientes al propio sistema. Esto significa que esperan a que estén disponibles los recursos y los retienen durante el tiempo que tardan en realizar la actividad. El control sobre la disponibilidad de los recursos se realiza mediante una tabla horaria. De esta manera se puede especificar cuándo estará cada recurso disponible y para desempeñar qué rol dentro del sistema. Por ejemplo, en un hospital los pacientes pasan de unos servicios a otros haciendo uso de los recursos del hospital. Algunos de estos recursos, como son los médicos, desempeñan funciones en diferentes unidades.

Teniendo en cuenta las características del problema, solamente existen dos circunstancias en las que cambia el reloj de simulación:

- un elemento realiza una espera voluntaria de tiempo (por ejemplo para simular el hecho de realizar una actividad una vez ha obtenido los recursos necesarios para la misma).

- un elemento trata de realizar una actividad y queda en espera (de tiempo no definido) de obtener los recursos necesarios.

La librería implementada pone a disposición del usuario los mecanismos necesarios para poder representar el paso del tiempo de simulación. Esto se consigue con un gestor del tiempo de simulación que funcionará como el motor que decide en cada momento que elementos del sistema deben estar activos para un tiempo de simulación y cuales están en proceso de espera. A este motor le llamaremos proceso lógico (PL). Los principales componentes de la librería son:

- Los Elementos que transitan por el sistema, realizando actividades y consumiendo tiempo de simulación ya sea haciendo esperas en tiempo o esperando por la obtención de recursos.

- Los Recursos del sistema, necesarios para que los Elementos puedan realizar sus actividades.

- El Proceso Lógico (PL) encargado de controlar el tiempo de simulación.

A continuación se describe la funcionalidad de la librería indicando cómo implementar las dos formas de hacer evolucionar el tiempo de simulación.

4. ESPERAS EN TIEMPO

En esta situación se simulan elementos que no realizan actividades de ningún tipo sino que su paso por el sistema se reduce únicamente a cambiar el estado del mismo con el paso del tiempo. En un sistema de eventos discretos, los cambios de estado se producen en instantes específicos de tiempo, por lo que el diagrama de actividad del elemento sería el que se indica en la figura 2.

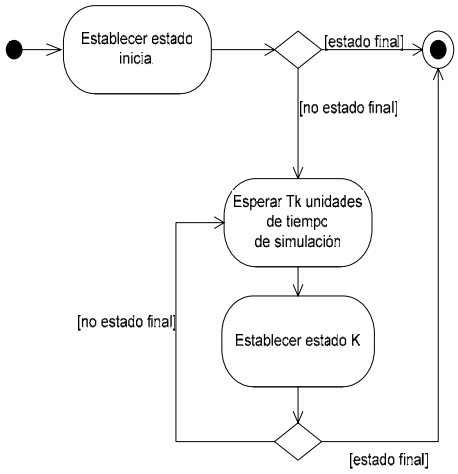

Figura 2. Diagrama de actividad de un elemento que realiza una espera en tiempo.

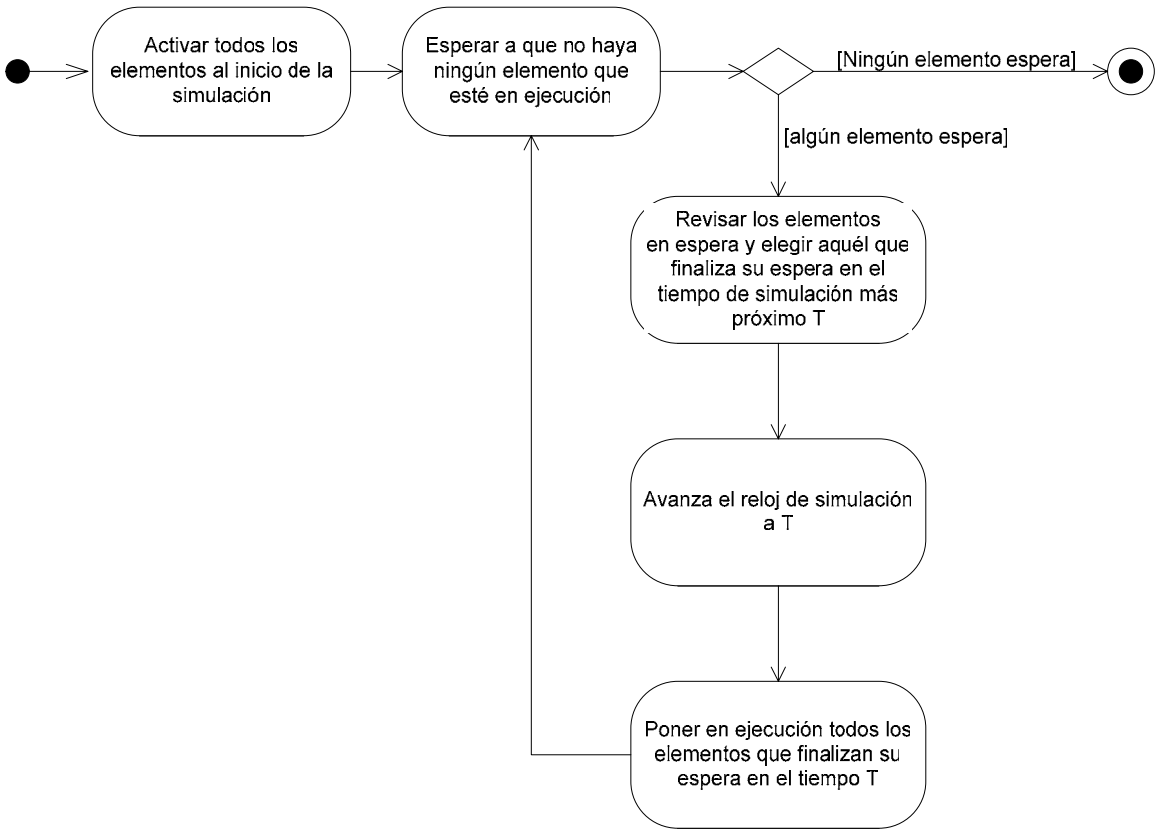

Figura 3. Diagrama de actividad del Proceso Lógico.

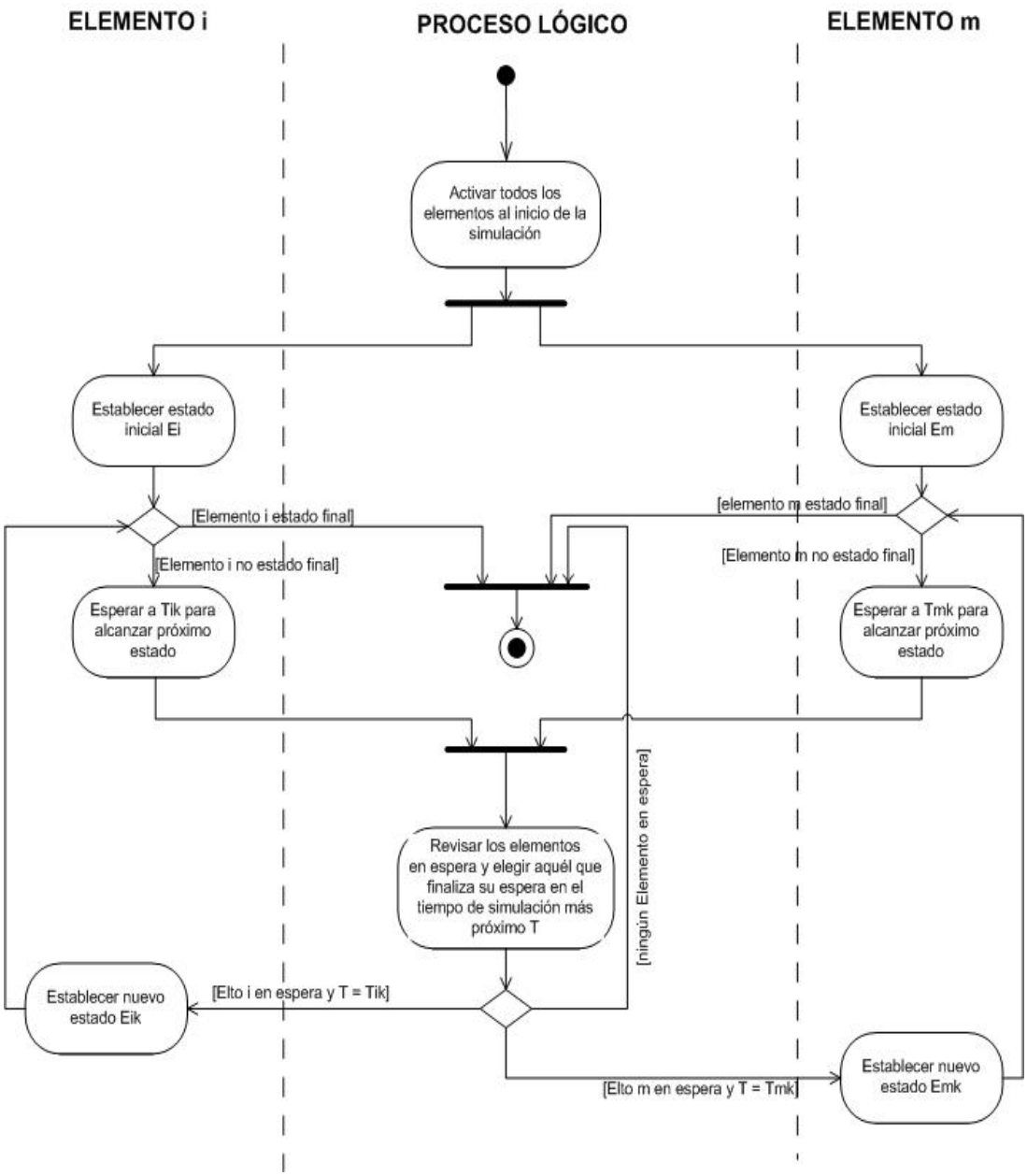

Figura 4. Diagrama de flujo entre los elementos y el proceso lógico.

Por otra parte, el gestor del tiempo de simulación (Proceso Lógico) se encarga de iniciar todos los elementos del sistema que tienen que estar en ejecución. Cuando estos elementos han finalizado la actualización de su estado significa que para ese tiempo de simulación ya no se van a producir más eventos que cambien el estado global del sistema. En este instante se tiene que revisar cual es el instante más próximo T en el que finalizará la espera en tiempo de cualquiera de los elementos que están interviniendo en el sistema, y una vez lo haya encontrado se avanza el reloj de simulación hasta este tiempo T. Una vez actualizado el reloj de simulación volverá a comenzar la ejecución de estos elementos que concluían su espera y volverá a esperar a que terminen de variar su estado para volver a repetir el proceso. Su diagrama de actividad se representa en la figura 3.

Tanto los elementos del sistema como el Proceso Lógico, se ejecutan como hilos independientes. Así pues, los elementos deben notificar al proceso lógico que ya han finalizado su actividad y que van a realizar una espera hasta un tiempo Tk. También el proceso lógico debe notificar a cada elemento k que debe despertarse cuando se haya alcanzado el tiempo de simulación Tk. En el diagrama de la figura 4 se observa este hecho.

Las características del Proceso Lógico serán:

- Debe tener un reloj que indique el tiempo actual de la simulación
- Debe tener almacenados los elementos que se están ejecutando para el tiempo de simulación que tenga marcado.

- Debe tener almacenados los elementos que están en espera de que se llegue a un determinado tiempo de simulación.

- Debe ser capaz de notificar a los elementos que se ha alcanzado el tiempo de simulación por el que estaban esperando.

Mientras que las de cada elemento son:

- Debe tener almacenado en qué tiempo de simulación debe volver a cambiar de estado

- Debe poderse identificar de forma única.

- Debe ser capaz de notificar al proceso Lógico que ha terminado su ejecución y va a realizar una espera, o bien a finalizar su función en el sistema.

5. REALIZAR ACTIVIDADES

Como se ha indicado anteriormente, los elementos evolucionan por el sistema realizando una serie de actividades y modificando su estado. La realización de una actividad conlleva tres acciones básicas:

- Obtener los recursos necesarios para desarrollar la actividad (compitiendo en su obtención con el resto de elementos).

- Modificar el estado del sistema, simulando el efecto que se produce cuando se realiza esta actividad, y hacer una espera en tiempo representativa de la duración de esta actividad en el sistema real (por lo que la realización de actividades lleva asociado realizar esperas en tiempo explicadas en el epígrafe anterior).

- Devolver los recursos utilizados tras realizar la actividad.

Los elementos solicitan recursos para utilizarlos. Si hay unidades disponibles de éstos recursos, entonces los adquieren. Si por el contrario no hubiera unidades suficientes para atender su petición, deben detener su ejecución hasta que se le puedan asignar los recursos solicitados. Nuevamente, será necesario tener mecanismos para detener la ejecución de un elemento y volverlo a reanudar.

Aparecen dos componentes imprescindibles en la librería: los Recursos y las Actividades. A continuación se expone la funcionalidad de la librería vista desde la perspectiva de los recursos y las actividades.

5.1. Los recursos

Para abarcar todos los posibles casos que se dan en los sistemas reales, se ha creado dos tipos diferentes de recurso: los Recursos Básicos y los Recursos Activos.

Los Recursos Básicos desempeñan un solo rol en el sistema, y no actúan según una tabla horaria preestablecida. Este tipo de recursos serán los que se empleen en el modelado de recursos que están disponibles el 100% del tiempo. Se caracterizarán por:

- Una descripción que los identifique.

- Un contador que indique el número de unidades disponibles del recurso.

Para utilizar este tipo de recurso, los elementos solicitan el recurso mediante un método de clase, indicando quién es el solicitante, con qué prioridad lo solicitan y el número de unidades requeridas. Si el recurso básico tiene unidades suficientes para poder atender la petición, ésta será servida llevando a cabo dos acciones: descontar las unidades de las disponibles, y almacenando la petición atendida para tener registro de qué elemento y en qué condiciones está usando actualmente el recuso. En el caso de que la petición no pueda ser atendida será almacenada en una cola de peticiones pendientes ordenadas por prioridad. Cuando un elemento termina de usar un recurso lo devolverá, lo que implica incrementar el número de unidades disponibles, eliminar la petición en uso del almacén correspondiente, y revisar las peticiones pendientes para ver si se pueden atender nuevas peticiones. El acceso para el uso de estos recursos básicos se debe hacer de forma sincronizada para evitar inconsistencias (ya que debe recordarse que los elementos se ejecutan como threads concurrentes).

Los Recursos Activos rigen su actividad según un horario. Un ejemplo de sistema que requiera el uso de éstos en su simulación podría ser un consultorio médico. Tanto la consulta (entiéndase como la habitación) como el resto de material posiblemente estén a disposición del médico durante todo el tiempo para poder atender a los pacientes (serían pues recursos básicos). Sin embargo, el médico no está de forma continuada realizando su labor. Éste atenderá a pacientes durante una serie de horas al día que podrán ser continuas en el tiempo o partidas según unos descansos. Dependiendo entonces de los tiempos de llegada de los pacientes, los tamaños de las colas variarán y por lo tanto puede cambiar el tiempo medio de espera de un paciente para ser atendido.

No es la disponibilidad horaria el único cambio que se introduce en esta nueva aproximación acerca del funcionamiento de los recursos. También es necesario poder reflejar que un mismo recurso puede actuar de diferentes maneras dentro del sistema, tal y como puede verse en el siguiente ejemplo: en un quirófano intervienen una serie de personas durante una operación, cada una con un papel determinado. Así, se tendrá un cirujano principal, uno o varios

ayudantes del cirujano y enfermeras. Cualquier cirujano puede actuar de ayudante en una operación, pero probablemente no todos los ayudantes puedan ejercer de cirujano principal debido a su falta de experiencia. Para poder modelar esta doble disponibilidad de un cirujano sería necesario definir dos recursos en el sistema, cuando en realidad se trata de un mismo recurso que juega dos papeles diferentes en función de las necesidades. Por ello, la librería permite definir recursos que actúan de diferente manera según un horario determinado. Diremos entonces que los recursos actúan como un tipo de recurso determinado, o actúan según un rol que no tiene que ser fijo durante toda la simulación.

La librería posee los mecanismos para definir los horarios de los recursos, y permite definir tablas horarias de casi cualquier tipo (no se ha forzado a que sean horarios semanales o mensuales). Una tabla horaria será una lista de tuplas con la siguiente estructura:
(ts_inicio, frecuencia, duracion, repeticiones, clase_recurso)

Esta tupla define de forma precisa los instantes de la simulación en los que el recurso activo se comportará según un rol. El significado de estos campos es el siguiente, un caso concreto se desarrollará en el apartado 6.1 de ejemplos (Paso 4):

- ts_inicio: Expresa en número de unidades de tiempo que han de transcurrir desde el inicio de un ciclo hasta que actúa bajo el rol.

- frecuencia: entero que expresa en número de unidades de tiempo que dura un ciclo.

- duracion: es un entero que expresa en número de unidades de tiempo durante las cuales va a actuar como el rol para esta entrada horaria.

- repeticiones: número de ciclos de esta entrada horaria que debe realizar durante la simulación. Un cero en este campo indica que el ciclo se repite de forma indefinida durante todo el tiempo de simulación.

- clase_recurso: referencia al rol que desempeña en esta entrada horaria.

Para darle aún mayor flexibilidad al uso de la librería se permite que el último campo de la tupla sea una lista de posibles roles que el recurso puede desempeñar durante esa entrada horaria. Con esta nueva funcionalidad se contempla el siguiente hecho: El médico “Luis Pérez” tendrá turno de consulta todos los jueves de ocho de la mañana a dos de la tarde. Sin embargo existe la posibilidad de que durante su turno de consulta sea llamado para realizar alguna intervención quirúrgica en urgencias. Sería entonces necesario poder definir la posibilidad de que durante ese horario desempeñara dos roles diferentes (dotando a la librería de mecanismos para no permitir que los realice de forma simultánea) “cirujano” y “médico de consulta”. El último campo de la tupla pasaría pues a ser una lista de roles.

La implementación del control del horario se ha basado en hilos de ejecución independientes (como la de los elementos) que activan o desactivan los recursos en función de los parámetros definidos en cada entrada horaria.

5.2. Las actividades

Una vez definidos cuáles son los recursos, es necesario modelar el conjunto de actividades que van a llevar a cabo los elementos que transitan por el sistema. Para que se puedan desarrollar estas actividades, se necesitará hacer uso de unos determinados tipos de recursos. La relación existente entre los recursos del sistema y las actividades que llevan a cabo los elementos viene definida de la siguiente manera: una actividad A necesitará de forma general de unos recursos de tipo TR1, … TRn. En un momento dado deberán existir recuros R1, … Rn que actúen como TR1, … TRn respectivamente. En este momento podrá realizarse la actividad A por el elemento E. Para ilustrar esto se presenta el siguiente ejemplo: Se dispone de un laboratorio donde se realizan dos tipos de pruebas, análisis de sangre y análisis de orina. Dos enfermeras manejan las máquinas de análisis: Laura, que trabaja desde las 8h hasta las 12h con la máquina de análisis de sangre y de 12h a 16h con la máquina de análisis de orina; y Paula, que sólo sabe manejar la máquina de análisis de sangre y lo hace de 12h a 16h. Si llega una petición de un análisis de orina a las 11h, éste no podrá llevarse a cabo hasta las 12h, ya que tiene que estar disponible un recurso que maneje la máquina de análisis de orina (y esto sólo se cumple cuando Laura desempeña ese rol de 12h a 16h).

El caso general de la ejecución de un elemento por el sistema sería realizar un grupo de actividades en un orden determinado. El elemento solicita la primera actividad, se pone en cola hasta obtener los recursos necesarios y una vez realizada, devuelve los recursos y solicita la siguiente actividad. Además otras situaciones que se pueden modelar con la librería son:

5.2.1. Realizar grupo de actividades sin orden secuencial. Con esta funcionalidad se modelan casos en los que un elemento tiene que realizar una serie de actividades que no tienen que ser ejecutadas en un orden determinado. La manera de operar será que el elemento se pondrá a la espera de poder realizar todas las actividades del grupo a la vez, sin tener que esperar la finalización de una para ponerse a la espera de realizar la otra. Un ejemplo sería: las revisiones médicas de una empresa se realizan en una mutua de seguros en la que los empleados deben pasar por consulta oftalmológica, por electrocardiograma, análisis de sangre y orina. Cuando los empleados de

la empresa llegan a la mutua solicitan hacerse la revisión, y la mutua con el fin de optimizar los recursos siempre está realizando pruebas. El primer empleado sería el primero de la cola en todas las listas, sin embargo, mientras se realiza la primera prueba, el segundo empleado, el tercero y el cuarto pueden empezar a hacerse el resto de pruebas. Cuando finalice la primera prueba, seguirá siendo el primero de la lista de espera en el resto de pruebas. Al final de la revisión, los empleados tendrá todas las pruebas realizadas, y de forma general, el primero que entró en la mutua será el primero que finalice, aunque no todos los empleados se han realizado las pruebas en el mismo orden.

5.2.2. Modelar actividades del sistema que se pueden realizar con diferentes tipos de recursos, variando su comportamiento en función del tipo de recursos escogido. El mejor modo de ver la aplicación de esta nueva funcionalidad vuelve a ser con un ejemplo. Sea un laboratorio que realiza análisis de sangre. El laboratorio dispone de dos máquinas, una bastante obsoleta y otra nueva que tarda la mitad de tiempo en realizar la misma prueba. Las peticiones de pruebas se van atendiendo por orden de llegada y se usa una máquina o la otra en función de su disponibilidad. Es necesario definir dos tipos de recursos diferentes (máquina_nueva y máquina_obsoleta) para tener en cuenta este hecho.

5.2.3. *Modelar actividades que no necesitan de la presencia del elemento para ser realizadas*. Dotando a la librería de esta posibilidad, la simulación podrá adaptarse a sistemas en los que los elementos tendrán la opción de solicitar determinados servicios que no requieran que el elemento solicitante espere por la finalización de los mismos, aunque como es obvio los recursos implicados en las actividades solicitadas queden ocupados con las mismas. Si un elemento de la simulación usa este mecanismo para hacer un grupo de actividades (A1,…,An) con duraciones (D1, …, Dn) y con la suposición de que no compite por los recursos con otros elementos, el tiempo que tardará en realizar el grupo de actividades será max(D1,…Dn).

Esta funcionalidad se podría confundir con la presentada en el punto 5.2.1. Sin embargo la diferencia es clara; en la 5.2.1. el elemento debe esperar la conclusión de una actividad para comenzar la siguiente (aunque ciertamente no importe el orden), mientras en la 5.2.3. la finalización de cualquier actividad en curso del elemento no es preceptiva para que pueda ir realizando otra. Como ejemplo tenemos un paciente en un hospital necesita hacerse una serie de pruebas médicas (análisis de sangre y citología). El paciente tendrá que esperar para hacerse las extracciones oportunas, sin embargo los resultados de ambas pruebas tardarán mucho más tiempo que las extracciones. El motivo es el siguiente; las actividades en este supuesto son las extracciones (actividades que se pueden realizar en grupo sin importar el orden como se explica en la 5.2.1) y el estudio de estas extracciones por parte de los especialistas son otras actividades, que no requieren la presencia del paciente y por tanto pertenecen a las que hemos clasificado en el punto 5.2.3.

6. EJEMPLOS

A continuación se presentan una serie de ejemplos de uso de la librería para simular diversas situaciones que pueden producirse en un hospital.

6.1. Actividades de petición simultánea

Cuando un paciente llega al hospital es habitual que se le pidan una serie de pruebas para poder obtener un diagnóstico. El paciente se pone en la cola de todas estas pruebas de forma simultánea ya que no le importa el orden en que se las realicen sino simplemente hacerlas todas. Por supuesto, el paciente no puede realizar más de una prueba simultáneamente.

En el ejemplo implementado se modela el servicio de Ginecología, dividido en cuatro tipos de patologías: Funcional, Orgánica, Oncológica y Precoz. Cada patología tiene una clase de consultorio y una clase de médico específicas.

Se dispone de dos tipos de pacientes que acuden a este servicio: los pacientes diagnosticados (PD), con un único tipo de patología y que requieren hacerse un número aleatorio de consultas sucesivas antes de obtener un diagnóstico; y los pacientes no diagnosticados (PND), que no saben exactamente qué padecen y tienen que pasar por todas las consultas.

Modelado del problema: El servicio de Ginecología se modela como un **Proceso Lógico** que gestionará el flujo diario de pacientes. Este flujo se creará mediante un generador de pacientes caracterizado como un **Elemento** que a las 0:00 horas de cada día genera un número prefijado de pacientes. La pertenencia del paciente al grupo de los PND o los PN se establece mediante una función de probabilidad.

Los pacientes se definen como **Elementos**. En el caso de los pacientes PN una función de probabilidad permite obtener el tipo de patología y el número de consultas sucesivas que deben pasar (la primera consulta es siempre obligatoria). Los pacientes PND pasan por las consultas de todas las patologías pero una sola vez.

Los tipos de médicos y consultorios se definen como **Clases de recursos**, una por cada patología: Médico Funcional, Médico Precoz, Consultorio Orgánica.... Cada médico y cada consultorio es un **Recurso Activo** con un horario asociado. En ese horario el médico realiza un único papel (o rol) como una de las clases de recursos definidas. Se dispondrá de esta

forma de un “Médico Funcional 1”, “Consultorio Orgánica 1”, etc.

Las consultas de cada una de las patologías se definen como Actividades que necesitan un único recurso de la clase médico y un único recurso de la clase consultorio de esa patología concreta.

Uso de la librería: Todo el servicio se describe dentro del Proceso Lógico mediante la sobreescritura del método crearContenido(), realizando los siguientes pasos:

- Paso 1: Se inicializan las actividades de las que se compone. Cada actividad se define mediante una descripción y una función de probabilidad para representar la duración de la actividad. En el primer caso es, por ejemplo, una normal de media 20.0 y desviación típica 5.0

```
Actividad actConsFuncional = new Actividad("Consulta Funcional", new Normal(20.0, 5.0));
```

- Paso 2: Se inicializan las clases de recursos, que simplemente necesitan una descripción

```
ClaseRecurso crMedFuncional = new ClaseRecurso("Médico Funcional");
```

- Paso 3: Se asocia a las actividades las clases de recurso correspondientes. Este paso permite caracterizar las actividades

```
actConsFuncional.anadeEntrada(crMedFuncional, 1);
actConsFuncional.anadeEntrada(crConsFuncional, 1);
```

- Paso 4: Hay que crear los recurso activos de cada clase y añadir sus entradas de horarios. En este caso tanto los médicos como los consultorios tienen un horario que comienza a las 8 de la mañana (480), cada día (1440), durante 6 horas (360).

```
RecursoActivo medFuncional = new RecursoActivo(this, "Médico F1");
 medFuncional.nuevaEntradaHorario(480, 1440, 360, crMedFuncional, 0);
```

Se implementa el Paciente como una extensión de la clase **Elemento**. Al paciente diagnosticado se le indica la consulta que tiene que solicitar, y se crea una lista con todas las consultas que tiene que solicitar para el paciente no diagnosticado. Finalmente, se implementa el programa principal como se muestra en la figura 5.

```
public class Hospital {
 static final int NDIAS = 60;
 static final int NPACIENTES = 20;

 public static void main(String arg[]) {

 // Se crea el PL estableciendo el tiempo de simulación
 Ginecologia gine=new Ginecologia(0.0,NDIAS*24*60);

// Se crea el Generador de paciente y se pone a ejecutar.
//El último parámetro es el porcentaje de pacientes PD
 Genera g=new Genera(0,gine,NDIAS,NPACIENTES,50);
 g.start();

  // Se arranca el proceso lógico
   gine.start();
  }
}
```

Figura 5. Código del programa principal.

Resultados: Este ejemplo y los siguientes se han ejecutado en un ordenador Pentium 4, 2,53 GHz, con 512 Mb de RAM. El sistema operativo utilizado ha sido Windows XP con la máquina virtual de Java versión 1.4.2_03. Los tiempos de ejecución obtenidos son los que se muestran en la figura 6 y como se puede observar la gestión de colas (generadas con 70 pacientes diarios) no aumenta excesivamente el tiempo de ejecución.

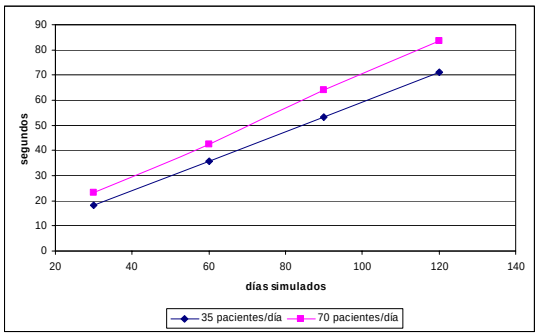

Figura 6. Tiempos de ejecución del ejemplo 1

6.2. Recursos activos con roles solapados en horario

En un hospital (como en cualquier sistema real) cada recurso puede usarse para más de una tarea, aunque no puede estar siendo usado por más de una tarea simultáneamente. Es el caso de las máquinas que obtienen una serie de resultados de las analíticas. Simplificando mucho podría verse que un hospital dispone de máquinas que realizan análisis de sangre y también de máquinas para los análisis de orina. Son dos máquinas distintas para dos actividades diferentes. Sin embargo, puede ocurrir que se disponga también de una máquina más moderna y cara capaz de realizar los dos tipos de prueba, aunque nunca simultáneamente. Se habla en este caso de una máquina que en el mismo horario puede realizar dos papeles (o roles) diferentes

Modelado del problema: En líneas generales es muy similar al del ejemplo anterior. El **Proceso Lógico** es simplemente un laboratorio de análisis. Se definen tres tipos de máquinas (**Clases de Recurso**): Máquina de análisis de orina, de sangre y de heces. Se dispone de cuatro máquinas (**Recursos Activos**) distintas: tres específicas, una para cada tipo de prueba; y una máquina capaz de realizar cualquiera de estas pruebas. Esta última máquina sólo puede realizar una de estas pruebas cada vez. Si fuese capaz de estar realizando las tres pruebas en el mismo instante de tiempo se podría modelar como tres máquinas distintas y no necesitaríamos usar esta característica de la librería. Las **Actividades** son cada una de las pruebas que puede realizarse un paciente: Análisis de sangre, de orina o de heces. Los pacientes (**Elementos**) realizan aleatoriamente una sola de las pruebas.

Uso de la librería: se implementa el código para crear el Proceso lógico siguiendo los mismos pasos del ejemplo anterior. La única modificación a destacar se produce en el paso 4, al asociarle el horario al recurso con rol múltiple en el mismo horario.

```
RecursoActivo completa1 = new RecursoActivo(this,
"Máquina Análisis Sangre, Orina y Heces");
ArrayList alCompleta1 = new ArrayList();
alCompleta1.add(crSangre);
alCompleta1.add(crOrina);
alCompleta1.add(crHeces);
completa1.nuevaEntradaHorario(480, 1440, 360,
alCompleta1, 0);
```

El código de los pacientes es muy parecido al de los pacientes diagnosticados del caso anterior pero sólo se realizan la prueba seleccionada una única vez. El programa principal también funciona exactamente igual que en el primer ejemplo.

Resultados: Se puede observar en la figura 7 los tiempos de ejecución para el caso de uso de recursos activos con roles solapados. Se observa que al aumentar el número de pacientes diarios se incrementa el tiempo de ejecución. Esto es debido a la gestión de horarios de los recursos, que es más compleja en el caso de roles solapados en horario.

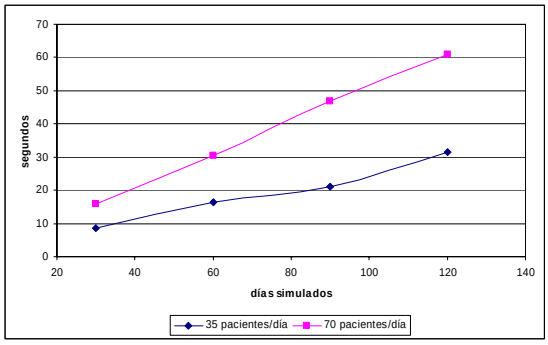

Figura 7. Tiempos de ejecución del ejemplo 2

6.3. Actividades no presenciales

El último ejemplo con el que se quiere ilustrar la potencia de la librería de simulación es un tipo de actividades de aparición habitual en un hospital. Hasta ahora se ha supuesto que las pruebas que se realiza un paciente requieren su presencia durante toda su duración. Esto, sin embargo, no es cierto si se quiere ser un poco más fiel al modelo real del hospital. Cuando un paciente necesita un análisis de sangre tan solo se le requiere durante la extracción de la muestra. Una vez obtenida ésta, se pasa a una máquina o un médico la estudia, pero no es necesario que el paciente esté presente mientras esto ocurre. El paciente puede emplear este tiempo incluso para realizar otras actividades, ya que, rigurosamente hablando, no está realizando ninguna actividad.

Las actividades no presenciales no tienen fijada a priori ninguna dependencia con el resto de actividades, como ocurre en el caso de las actividades "presenciales". Por ello la librería contiene mecanismos para forzar al sistema a esperar por las actividades no presenciales antes de continuar su ejecución.

Modelado del problema: Los pacientes se modelan como un tipo especial de Elemento, los **ElementosNP** (No Presenciales), para permitirle solicitar actividades no presenciales. Se han definido tres **actividades** distintas: una presencial, la toma de muestras del paciente (extracción de sangre y muestra de orina); y dos no presenciales que representan el análisis de cada uno de los tipos de muestra. Para realizar estas actividades se requiere en todos los casos un enfermero y dos tipos de máquina: para analizar la sangre y para analizar la orina. Estos son las tres **Clases de Recurso** definidas y los **Recursos Activos** correspondientes.

Uso de la librería: La única diferencia en la implementación con los casos anteriores está en el paso 1 donde se inicializan las actividades. Las actividades no presenciales requieren un último parámetro:

```
Actividad actAnaSangre = new Actividad("Análisis de
sangre", 0, new Normal(10.0, 2.0), false);
```

Para ver las posibilidades de uso de las actividades no presenciales se define el flujo del paciente como se describe en la figura 8.

```
public void run() {
 Actividad a;
 a = pl.getActividad("Extracción sangre, muestra
orina");

 try {
    realizaActividad(a);

   // Comienza las actividades no presenciales
   Actividad aNP1 = pl.getActividad("Análisis
sangre");
   Actividad aNP2 = pl.getActividad("Análisis
orina");
   ArrayList actividades = new ArrayList();
   actividades.add(aNP1);
   actividades.add(aNP2);

    realizaActividades(actividades);

   // Se pone un punto explícito de espera por las
actividades no presenciales
   esperaNP();

   // Y el paciente realiza otra vez todo el proceso
   realizaActividad(a);

   finaliza();
   } catch(ExcepcionFinSimulacion ef) {
   printMsg("<< Fin del tiempo de simulacion >>");
   } finally {
  finaliza();
  }
}
```

Figura 8. Código del elemento paciente que realiza actividades no presenciales.

Resultados: No se observan diferencias en los tiempos de ejecución cuando la llegada diaria de pacientes varía entre 35 y 70. Esto es debido a que por cada actividad no presencial se crean nuevos hilos de ejecución y las colas se llenan en ambos casos. Si la llegada diaria es de 8 pacientes, no se crean colas y se ve que el tiempo de ejecución es inferior, figura 9.

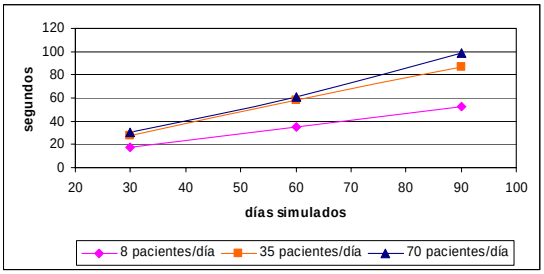

Figura 9. Tiempos de ejecución del ejemplo 3.

7. CONCLUSIONES

En este artículo se ha descrito una librería Java para la simulación de sistemas de eventos discretos. Aunque inicialmente está desarrollada para su uso en la gestión hospitalaria, se ha diseñado de manera que se pueda modelar cualquier sistema similar siguiendo una metodología orientada al proceso. El cuello de botella de este tipo de simulación es el número de elementos activos debido al número máximo de hilos que se pueden ejecutar simultáneamente en la máquina virtual Java. El límite de días de simulación va a depender tanto del número de pacientes diarios como de su flujo y tipo de actividades a realizar. El uso de recursos solapados en horarios también empeora el rendimiento del programa. Para resolver estas restricciones se está trabajando en la paralelización de la librería.

AGRADECIMIENTOS

Este trabajo ha sido cofinanciado por el Ministerio de Ciencia y Tecnología y Fondos FEDER, a través del proyecto ”SIGHOS: La Simulación Inteligente en la Gestión de recursos Hospitalarios” con nº de referencia DPI 2003-04884. Los autores también desean agradecer el apoyo recibido por el Hospital Nª Sª de la Candelaria (S/C de Tenerife) y especialmente, por D. Carlos Bermudez Pérez.

REFERENCIAS

Banks J., Carson J.S., Nelson B.L.(2001), *Discrete-Event System Simulation*, Prentice-Hall

Barber P (1993), *La simulación como herramienta de Gestión: una aplicación al ámbito sanitario*, Tesis Doctoral, Universidad de Las Palmas de Gran Canaria

Barceló J., Casas J.,Ferrer J.L. and García D. (1999a), Modeling Advanced Transport Telematic Applications with Microscopic Simulators: The case of AIMSUN2, in: Traffic and Mobility, Simulation, Economics, Environment, W. Brilon, F. Huber, M. Schreckenberg and H. Wallentowitz (Eds.), Springer

Barceló J., Ferrer J.L. and Martín R. (1999b), Simulation Assisted Design and Assessment of Vehicle Guidance Systems, *International Transactions in Operations Research*, Vol. 6, No.1, pp. 123-143

Barceló J., Casas J. and Ferrer J.L. (1999c), Analysis of Dynamic Guidance Systems by Microsimulation with AIMSUN2, *Proceedings of the 6th World Conference on ITS*, Toronto

CACI Products Company (1995), *Modsim II. The Language for Object-Oriented Programming. Reference Manual.*

Fishwick P.A. (1995), *Simulation Model Design and Execution. Building Digital Worlds*, Prentice Hall.

Gilbert G.N., Doran J (1993), *Simulating societies: the computer simulation of social processes*, UCL Press.

González B., Barber P. (1994), Evaluation of Alternative Functional Designs in an Emergency Department by Means of Simulation”, *SIMULATION*, **63:1**, pp 20-28

Guasch A, Piera M., Casanovas J., Figueras J. (2002) *Modelado y Simulación. Aplicación a procesos logísticos de fabricación y servicios*. Edicions UPC.

Langton, Christopher, Minar, Burkhart, *The Swarm Simulation System: A Tool for Studying Complex Systems*, Santa Fe Institute, www.swarm.org.

Magee J, Kramer J (1999), *Concurrent Programming in Java: Design Principles and Patterns*, 2nd Ed. John Wiley

MedModel Home Page (2005), http://www.promodel.com/products/medmodel/

Moreno L., Aguilar R.M., Martín C.A., Piñeiro J.D., Estévez J.I.,Sigut J.F., J.L. Sánchez (2000), Patient-Centered Simulation to Aid Decision-Making in Hospital Management, *Simulation*, vol.74, nº5

Pidd M. (1998), *Computer Simulation in Management Science*, Wiley

SIGHOS Home Page (2005), http://simula.cyc.ull.es/