XXV Jornadas de Automática
Ciudad Real, del 8 al 10 de septiembre de 2004

LIBRERÍA EN JAVA PARA LA SIMULACIÓN DE SISTEMAS DE EVENTOS DISCRETOS

R.M. Aguilar
Dpto. Física F.E., Electrónica y Sistemas. Universidad de La Laguna, raguilar@ull.es

C. Martín, I. Castilla, V. Muñoz, L. Moreno
Dpto. Física F.E., Electrónica y Sistemas. Universidad de La Laguna, cmartin@iac.es

Resumen

El objetivo general en el que se enmarca este trabajo es la creación de una herramienta que ayude en la toma de decisiones de la gerencia hospitalaria. Para ello se pretende implementar un paquete de Simulación que junto con técnicas de Inteligencia Artificial ayuden a tener un mejor conocimiento de los procesos hospitalarios. En éste artículo se describe la librería, implementada en Java, para la simulación de un sistema de eventos discretos. Este paquete pretende ser un conjunto de funciones que ayuden al ingeniero en la implementación de la simulación de un sistema de eventos discretos, abstrayéndose de los detalles técnicos de bajo nivel comunes en todas las implementaciones de este tipo.

Palabras Clave: Simulación, Sistemas de Eventos Discretos, JAVA, modelado orientado al proceso.

1 INTRODUCCIÓN

A medida que el mundo avanza, se crece en tecnología permitiendo abordar problemas cada vez más complejos; sin embargo, las exigencias del mercado obligan a tener márgenes de error muy pequeños. En cualquier toma de decisiones, ya sea en empresas, en la industrias o en las organizaciones gubernamenteles, se tiene que tener un conocimiento muy profundo del problema a tratar y no se pueden tomar acciones de control en base a suposiciones. Se exige, por lo tanto, técnicas de análisis fiables que minimizen los costes producidos por errores en la toma de decisiones. Una de estas técnicas es la simulación por ordenador.

Pero, ¿qué es la simulación computacional? La simulación es simplemente el uso de un modelo para imitar el comportamiento de los sistemas y permitir estudiar sus rendimientos bajo una variedad de circunstancias. La simulación es frecuentemente utilizada para determinar algunos aspectos del sistema. Por ejemplo, si quisiéramos saber cómo el número de trabajadores en un banco afecta al rendimiento del mismo. El primer paso sería la construcción del modelo que replique la llegada y atención de clientes. Este modelo podría usar variables aleatorias para generar cantidades tales como número de clientes, tiempo de llegada de los clientes, tiempo de atención a los clientes, etc. El siguiente paso sería hacer evolucionar el modelo, esto es, realizar la simulación del proceso. A continuación se estudian los datos obtenidos acerca de los clientes que se han atendido y se determinan cuántos trabajadores necesitamos para tener máximo rendimiento (mayor número de cliente atendidos al menor coste posible, esto es, con el menor número de trabajadores). La simulación por ordendor es definida de la misma manera, con la variación de que el modelo tiene que ser un programa de ordenador.

En los últimos años la simulación computacional se ha convertido en una de las principales técnicas en el estudio de sistemas complejos. Esto es debido a que los ordenadores son cada vez más baratos y con mayores prestaciones, así como la mejora en las herramientas para la simulación por computador, que han hecho que se puedan estudiar sistemas complejos.

Con la tendencia de aplicar el modelado y simulación al estudio de sistemas complejos o sistemas a gran escala, las investigaciones se han dirigido a explorar las posibilidades de generar algoritmos de simulación con un buen funcionamiento. El cambio presentado en este campo es el desarrollo de herramientas de simulación eficientes y fiables que no supongan una gran carga, a los ingenieros, en los detalles técnicos de bajo nivel cuando realizan el modelo. Esta es la línea del trabajo que se describe 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. Esta librería está siendo diseñada para soportar tanto simulación secuencial como paralela. La sincronización de los eventos y los detalles de la gestión de la programación son transparentes al usuario. Esto permite que el ingeniero se centre en el

modelado y la traducción del modelo conceptual del problema.

2 EL MODELADO DE SISTEMAS DE EVENTOS DISCRETOS

Como hemos comentado anteriormente, la simulación es una disciplina esencial en el estudio de sistemas complejos. Es útil disponer de una herramienta que te permita analizar los efectos y potenciales de las soluciones a problemas del mundo real. Muchos de estos casos se pueden abordar realizando un modelo de eventos discreto, esto es, un modelo dinámico, estocástico y discreto a intervalos no regulares de tiempo [1].

Como su nombre indica, en la simulación de eventos discretos el siguiente evento indicará el comportamiento del modelo. Muchas aplicaciones de simulación de eventos discretos utilizan un sistema de cola de una u otra clase. La estructura de cola puede ser tan obvia como una cola de trabajos esperando a ser procesados por una máquina, o una cola de aviones esperando a aterrizar. En otros casos esta cola no es tan evidente como una situación de incendios en una gran ciudad, donde los clientes son los fuegos que estan esperando a ser atendidos por los bomberos (servidores). [6]. En un sistema de eventos discreto existen dos tipos fundamentales de elementos, la entidad y el recurso.

Las entidades son los elementos individuales del sistema que estan siendo simulados y cuyo comportamiento está siendo ejecutado. Ej. máquinas en una factoría, pacientes en un hospital o aviones en un aeropuerto. En la simulación, el programa mantiene información de cada entidad y por lo tanto cada una puede ser individualmente identificada. Cuando una entidad cambia de estado en la simulación, el programa actualiza este nuevo estado. El estado del sistema global es el resultado de la interacción de las entidades individuales.

Mientras que los recursos son elementos individuales del sistema pero que no son modelados individualmente. Ellos son tratados como un conjunto de elementos cuyo comportamiento individual no es mantenido por el programa. Ej. el nº de cajas de un producto que hay en un almacen. El recurso consiste de un número de elementos identicos, por ello el programa mantiene el nº de recursos disponibles pero no su estado individual. Si un elemento del sistema debe ser tratado como entidad o recurso es una cuestión a resolver por el modelador.
En el desarrollo de una simulación de eventos discretos se necesita ver el sistema desde el punto de vista que ayude en el proceso de modelado. Todos los software de simulación disponibles usan fundamentalmente una de las siguientes dos aproximaciones: orientada al evento (redes de Petri) y orientada al proceso (ModSim II). En un modelado orientado al evento es necesario identificar todos los eventos diferentes que puedan ocurrir en el sistema y determinar que efectos producen esos eventos. Un evento se define como una ocurrencia instantánea que puede cambiar el estado del sistema. Mientras que en la aproximación orientada al proceso lo que se define es la secuencia de pasos (eventos o serie de eventos) para cada transación que ocurre en el sistema, esto es, se define cada proceso que sucede en el sistema. Considerando que un proceso es una secuencia ordenada en el tiempo de eventos interrelacionados separados por el paso del tiempo. Esta secuencia describe el paso de un item (individuo, información, ...) a través del sistema.

Las librerías que proponemos para el modelado y simulación de sistemas de eventos discretos son implementadas considerando el modelado orientado al proceso.

3 JAVA COMO LENGUAJE DE PROGRAMACIÓN

Java es simultáneamente una plataforma y un lenguaje de programación orientado a objetos diseñado por Sun Microsystems.

A continuación pasamos a describir las carácteríticas del Java que nos han llevado a utilizarlo como lenguaje de programación de las librerías de simulación de sistemas de eventos discretos.

3.1 JAVA ES SIMPLE

La sintaxis de Java es similar a la del lenguaje C o del C++, pero omite aquellas características semánticas que los hacen complejos, confusos y poco seguros: imposibilidad de que el programador gestione punteros, falta de herencia múltiple, de sobrecarga de tipos ni de operadores, ausencia de macros o de archivos de encabezados (.h en el lenguaje C). Además, Java dispone de un sistema llamado recogida de basura (garbage colector), que se ocupa de la destrucción automática de los objetos que ya no se utilizan, con el fin de liberar memoria. Java permite la gestión de excepciones (errores de ejecución).

3.2 UN LENGUAJE ORIENTADO A OBJETOS

Las ventajas de la programación orientada a objetos son: un mejor dominio de la complejidad (dividiendo un problema complejo en una serie de pequeños problemas) y una reutilización más simple así como mejores correcciones y evoluciones. Java está provisto de un conjunto de clases que permiten manipular todo tipo de objetos (interfaz gráfica, acceso a la red, gestión entrada/salida, ...).

3.3. DISTRIBUIDO

Java se distribuye con los protocolos de acceso a la red, como TCP/IP, UDP, HTTP o FTP. Esto permite realizar desarrollos en arquitectura cliente/servidor, para acceder a los datos de una máquina remota. Con lo cual, se puede desarrollar aplicaciones para acceder a información en la red, de la misma forma que se hace para accedera a los discos locales.

3.4. INTERPRETADO

Un programa Java no se ejecuta, es interpretado por la máquina virtual o JVM (Java Virtual Machine), lo que hace que no sea necesario recompilar un programa Java para cambiarlo de sistema, siendo tan solo necesario poseer la máquina virtual Java propia de cada uno de los sistemas.

Pero además, para que sea un lenguaje totalmente independiente de la máquina y del sistema operativo, debe ser compilado e interpretado. El código intermedio que se genera al ser compilado, llamado bytecodes, es código máquina de muy bajo nivel, y corresponde al 80% de las instrucciones del programa. Para adaptar este código genérico a una plataforma concreta (un procesador y un sistema operativo concreto), debemos interpretarlo para añadir el 20% que falta para su ejecución. En la figura 1 podemos ver los pasos que sique un programa Java hasta ejecutarse en la máquina.

Evidentemente, el carácter interpretado del lenguaje Java influye en su rendimiento. Aunque es mejor que un lenguaje de *script*, y lo suficientemente rápido para aplicaciones interactivas, Java es más lento que los lenguajes tradicionales cuyo código fuente es compilado directamente en el código máquina de una plataforma concreta. Para mejorar el rendimeinto del Java, y solventar este problema, se ha desarrollado el compilador "justo a tiempo" (Just-In-Time, JIT) que se ejecuta concurrentemente con la máquina virtual de Java. JIT determina que partes del código se ejecutan más frecuentemente para compilarlas dinámicamente en código ejecutable y que no tengan que ser interpretadas. Esto es, cuando en una máquina virtual de Java hay un compilador JIT, después de traerse una clase y verificarla, y antes de ejecutarla, se compila al código máquina nativo. Este proceso de compilación es gradual, es decir, se realiza conforme se van necesitando las clases. El rendimiento de un programa en Java es comparable al de uno realizado en C [3].

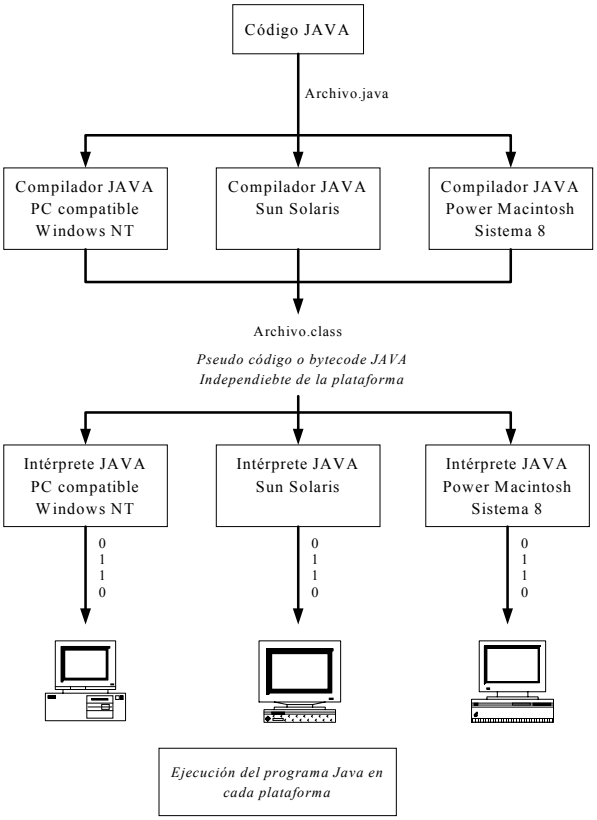

Figura 1: Esquema de concepción de un programa Java

3.5. ROBUSTO

Java incorpora toda una serie de medidas y comprobaciones para evitar errores inesperados. Realiza chequeos tanto en tiempo de compilación, como en tiempo de ejecución. También comprueba los tipos de datos y obliga a una definición explícita de los métodos que se definen para las clases en Java. Además, la gestión de los punteros es realizada en su totalidad por Java, sin que el programador tenga medio alguno de acceder a ella, lo que evita la posibilidad de sobreescribir datos en memoria de forma inoportuna.

3.6 SU SEGURIDAD ES MUY ALTA

La seguridad de Java incorpora 4 niveles que deben servir para controlar, entre otras, la posibilidad de ejecutar software de manera distribuida entre diferentes máquinas, y de cargar applets (programas en Java que se carga en un navegador) procedentes de internet. Para ello, el código es verificado en la compilación, y también por el intérprete en el

momento de la ejecución. Los niveles de seguridad son:

- Nivel de lenguaje: se realiza una gestión automática de la memoria y se eliminan los punteros, con ello se reduce al 50% los errores producidos durante la ejecución de un programa.
- Nivel de verificación de los bytecode: Java realiza una comprobación de los bytecode en busca de irregularidades y detectando posibles anomalías.
- Nivel de cargador de clases: cuando se carga una clase utilizada por una aplicación, estas serán sometidas a una serie de reconocimientos y comprobaciones, estén en el ordenador local, distribuidas en una red local o en Internet.
- Nivel de API de Java: Java incluye mecanismos de control y comprobación sobre las clases de acceso a los recursos del sistema

La seguridad para Internet se centra en la ejecución de los applets. La posibilidad de cargar un applet desde cualquier punto de la red, es una puerta abierta para virus y programas no deseados. Por ello, todos los navegadores actuales, a partir de 1998, incorporan una serie de restricciones: los applets no pueden ejecutar programas sobre el equipo local, tampoco pueden leer o escribir sobre este equipo y sólo se podrán conectar al servidor que los contenía antes, es decir, de donde fueron descargados.

3.7 INDEPENDIENTE DE LAS ARQUITECTURAS HARDWARE

El lenguaje Java ha sido diseñado para crear aplicaciones que funcionen en entornos de red, operando en una amplia variedad de arquitecturas hardware y sistemas operativos. El compilador de Java genera bytecodes, un formato intermedio independiente de la arquitectura utilizado para transportar el código entre los distintos sistemas hardware y software. El bytecode es el código máquina para la Máquina Virtual Java, que es el procesador virtual para el que están generados los programas Java. Debido a la naturaleza interpretada del lenguaje, el mismo bytecode puede ejecutarse sobre cualquier plataforma sin necesidad de recompilación, figura 1.

3.8. PORTABLE

La neutralidad respecto a la arquitectura es sólo una parte de un sistema verdaderamente portable. El lenguaje Java va más allá, definiendo estrictamente las especificaciones del lenguaje. En Java están definidos, por ejemplo, el tamaño de los tipos de datos básicos (norma IEEE754) o el comportamiento estricto de todos los operadores aritméticos sea cual sea la plataforma de desarrollo.

Los programas se ejecutan sobre la Máquina Virtual Java, que especifica todas las instrucciones permitidas y su significado. Para que los bytecodes se puedan ejecutar sobre un nuevo sistema hardware, sólo es necesario portar la máquina virtual a ese nuevo sistema, y todos los programas existentes pasaran a ejecutarse sin problemas

3.9 DINÁMICO

La naturaleza portable e interpretada de Java proporcionan un sistema dinámico y dinámicamente extensible. Las clases son enlazadas en el momento en que son requeridas, no antes de la ejecución del programa, y pueden ser cargadas de la red. Todo el código nuevo es verificado antes de ser ejecutado.

La carga dinámica permite evitar tener que recompilar todos los programas que contengan clases hijas cuando se modifica una clase padre. También permite actualizar un programa simplemente cambiando algunas de sus clases, sin tener que recompilarlo completo, siempre que las nuevas clases mantengan la misma interfaz o un superconjunto.

3.10 MULTITAREA

Java permite desarrollar aplicaciones que realicen la ejecución simultánea de varios threads (procesos ligeros o hilos de ejecución). Y este soporte lo da a nivel del lenguaje de programación, y no como unas librerías añadidas aparte, como es normal en otros lenguajes, lo que le da al lenguaje una mayor robustez y sencillez de uso.

Existe una clase `Thread` a partir de la cual pueden derivarse nuevos objetos thread. Cada una de estas threads es un hilo de ejecución distinto dentro del mismo proceso. La clase Thread proporciona métodos para crear una nueva thread, ejecutarla, detenerla y conocer su estado. También se incluyen primitivas de sincronización basadas en el empleo de monitores y variables condición. Todas las clases que forman la librería estándar de Java han sido diseñadas de modo que puedan ser utilizadas simultáneamente por varias threads. Cualquier objeto puede ser convertido en una thread y comenzar su ejecución por separado con un mínimo de esfuerzo de programación

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

La librería que queremos desarrollar trata de proporcionar 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 de la misma y sobrescribiendo aquellos métodos que fueran necesarios para representar las características particulares del mismo. Decidiría entonces los parámetros de la simulación (entradas y tiempo que se quiere simular) y obtendría los resultados deseados al final de la misma. Y todo ello, abstrayéndose de la engorrosa tarea de programar cada una de los procedimientos utilizados de forma genérica en una simulación de eventos discretos.

La metodología de modelado usada es la orientada al proceso, por lo que el sistema se caracterizará porque habrá una serie de elementos que van discurriendo por una serie de etapas que pueden ser unas u otras en función del estado del propio elemento. Dentro de cada etapa estos elementos realizarán diferentes actividades. Para poder efectuarlas harán uso de recursos pertenecientes al propio sistema. Esto significa que esperarán a que estén disponibles los recursos y los retendrán durante el tiempo que tarden en realizar la actividad. El control sobre la disponibilidad de los recursos se realizará mediante una tabla horaria. De esta manera se podrá especificar cuándo estará cada recurso disponible y para desempeñar qué actividad dentro del sistema. Como ejemplo podríamos mencionar cualquier sistema hospitalario, en el que los pacientes van pasando 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. Éste mismo ejemplo es aplicable en muchos sistemas sociales [4].

Es importante el concepto de "tiempo de simulación", que será el tiempo que queremos emular el comportamiento del sistema en estudio. Para permitir que la secuencia de eventos relacionados con el tiempo ocurra se debe crear un "reloj de simulación". Este reloj funciona como un índice cronológico o planificador de instrucciones que indica cuando ciertos eventos deben ocurrir.

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.

Por lo que 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). Para ello hemos implementado tres clases de objetos básicos en la libreríra:

- Clase Elemento para representar los elementos básicos que transitan por nuestro sistema, y que serán capaces de usar recursos y consumir tiempo de simulación
- Clase Recurso para representar los objetos necesarios para que se desarrollen las actividades del sistema
- Clase Proceso Lógico que controlará el tiempo de simulación.

Para comprender mejor el funcionamiento y la relación entre estas clases de objetos vamos a hacer una exposición por partes, añadiendo funcionalidades a la librería de forma progresiva.

4.1 PRIMERA APROXIMACIÓN. LOS ELEMENTOS Y LAS ESPERAS DE TIEMPO.

Como primera aproximación supongamos que tratamos de simular sistemas en la que los elementos no necesitan de recursos para realizar su actividad. Su paso por el sistema se reduce únicamente a cambiar el estado del mismo con el paso del tiempo. Como estamos centrados en simulación de eventos discretos, los cambios en el estado se producen en instantes específicos de tiempo. Su diagrama de actividad sería se muestra en la figura 2.

Por otra parte, el gestor del tiempo de simulación, que hemos llamado Proceso Lógico se encargará de iniciar todos los elementos del sistema que tienen que estar en ejecución para adquirir su estado inicial y esperar a que todos hayan finalizado de establecerlo. Cuando esto ha ocurrido significará que para ese tiempo de simulación ya no se van a producir más eventos que cambien el estado global del sistema. En este instante tendrá que revisar cual es 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 deberá avanzar el reloj de simulación hasta este tiempo T. Una vez avanzado el reloj de simulación volverá a iniciar la ejecución de estos elementos que finalizaban su espera y volverá a esperar a que terminen de variar su estado para volver a repetir el proceso. Su diagrama de actividad sería el representado en la figura 3.

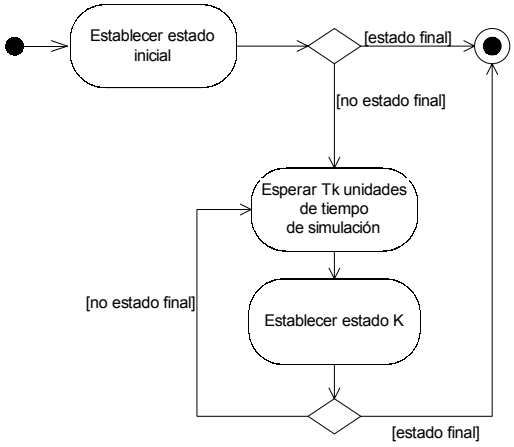

Figura 2: Diagrama de actividad de la clase elemento.

Por otra parte, el gestor del tiempo de simulación, que hemos llamado Proceso Lógico se encargará de iniciar todos los elementos del sistema que tienen que estar en ejecución para adquirir su estado inicial y esperar a que todos hayan finalizado de establecerlo. Cuando esto ha ocurrido significará que para ese tiempo de simulación ya no se van a producir más eventos que cambien el estado global del sistema. En este instante tendrá que revisar cual es 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 deberá avanzar el reloj de simulación hasta este tiempo T. Una vez avanzado el reloj de simulación volverá a iniciar la ejecución de estos elementos que finalizaban su espera y volverá a esperar a que terminen de variar su estado para volver a repetir el proceso. Su diagrama de actividad sería el representado en la figura 3.

Todos los elementos del sistema que estamos simulando, así 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 un 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. Para reflejar este hecho podemos ver los diagramas de forma conjunta (intervienen solamente dos elementos y el proceso lógico) en la figura 4.

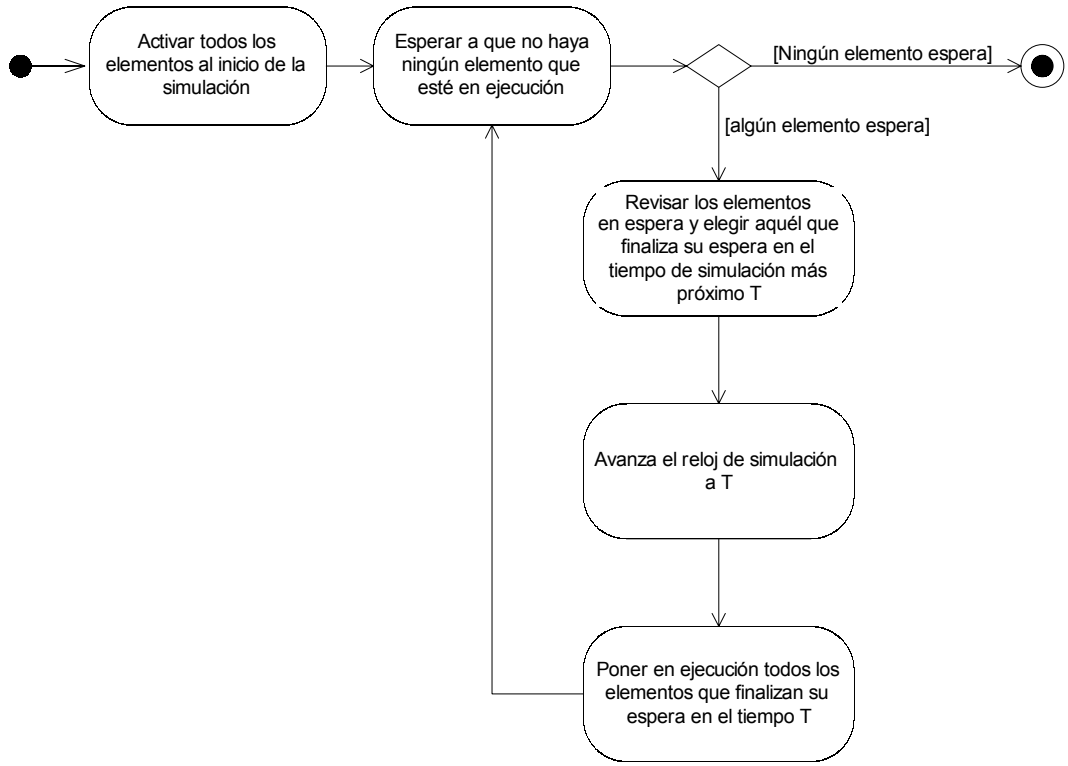

Figura 3: Diagrama de actividad del Proceso Lógico (PL).

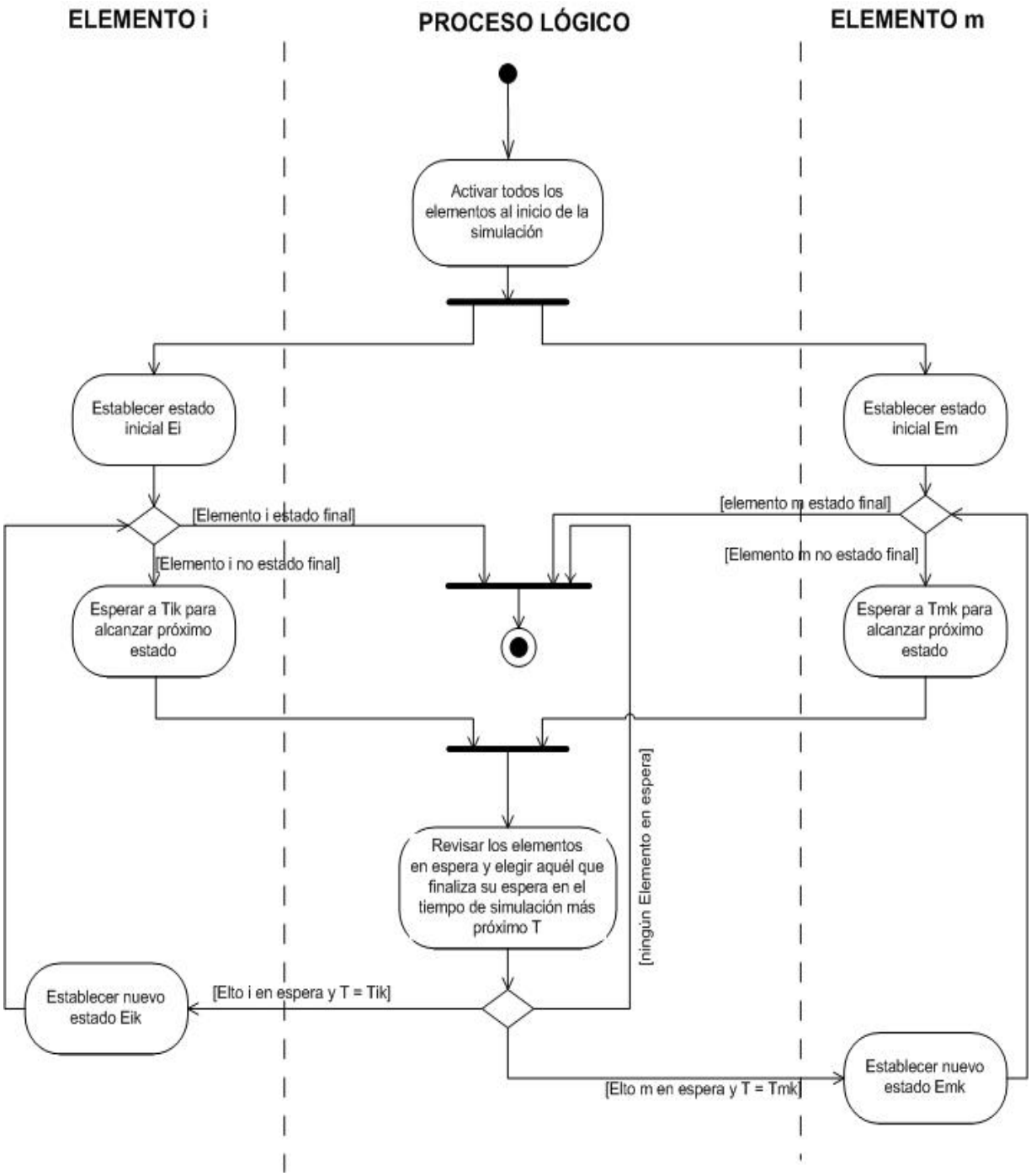

Figura 4: Diagrama de flujo entre los elementos y el proceso lógico.

4.2 CARACTERÍSTICAS DE LOS PROCESOS LÓGICOS Y DE LOS ELEMENTOS

A partir de los diagramas de flujo entre los elementos y el PL, podemos obtener las siguientes características.

El Proceso Lógico tiene que tener un reloj que indique en que tiempo va la simulación. Tiene que almacenar tanto los elementos que están ejecutando para el tiempo de simulación que tenga marcado, como los elementos que están en espera de que se llegue a un tiempo de simulación futura. Además, debe ser capaz de notificar a los elementos que se ha alcanzado el tiempo de simulación por el que estaban esperando.

Asimismo, cada elemento debe tener almacenado en que tiempo de simulación debe volver a cambiar de estado, debe poderse identificar de forma única y ser capaz de notificar al PL que ha terminado su

ejecución y va a realizar una espera, o bien a finalizar su función en el sistema.

Para una mayor eficiencia, se guardan los elementos que están en la cola de espera, en un montículo. El motivo de hacerlo así, es que cuando el proceso lógico detecta que ya no hay ningún elemento en ejecución debe sacar de la cola de espera, aquél que tiene su atributo de *ts* (tiempo de simulación en el que debe reanudar su ejecución) más pequeño, así como todos los demás de la cola que tenga el mismo valor de *ts* que el primero que ha extraído para ponerlos a ejecutar. Así, las inserciones en la cola de espera serán de O(log(n)) a diferencia de las inserciones en una lista ordenada que serían de O(n), siendo n el número de elementos que están en la cola de espera. El paquete que contiene las clases para implementar el montículo fue recogido de la referencia [5].

La cola de ejecución por otro lado únicamente tiene la función de almacén de elementos que en ese tiempo de simulación deben estar ejecutando, siendo suficiente con una estructura de datos tipo vector. Con todo ello el diagrama de clase se queda como se muestra en la figura 5. Un proceso Lógico será el encargado de gestionar los tiempos y las ejecuciones de un grupo de elementos (aquellos que usen los recursos administrados por él). Un elemento sin embargo, únicamente podrá ser controlado por un proceso lógico en cada momento.

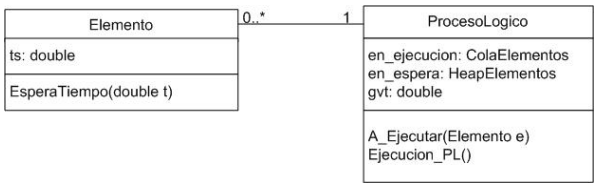

Figura 5: Diagrama de clase

4.3. MECANISMO DE SINCRONIZACIÓN

Como se ha mencionado anteriormente, los elementos se ejecutan como threads independientes en el sistema. Hemos visto que cuando hacen una espera en tiempo de simulación, su ejecución debe detenerse, para posteriormente ser reanudada por el proceso lógico, una vez que el reloj global de simulación haya avanzado hasta el tiempo de espera de ese elemento. Asimismo, el proceso lógico, tras haber lanzado a ejecutar los elementos de un tiempo de simulación concreto, debe detener su ejecución hasta que todos ellos realicen una espera en tiempo. Es evidente pues, que debemos incluir en nuestras librerías unos mecanismos de sincronización entre hebras, de manera que una hebra pueda detenerse y reanudarse de forma segura (obviamente sin usar esperas activas que consumen tiempo de ejecución).

Básicamente lo que se pretende es implementar dos métodos, uno para detener la ejecución de una hebra y otro para reanudar la ejecución de hebra (recordemos que todos los elementos del sistema van a ejecutar como hebras independientes). Java (y ese es otro de los motivos de elección de este lenguaje de programación para nuestras librerías) tiene implementado algunos mecanismos de sincronización entre hebras que a continuación se exponen:

- Cada objeto que tengamos en una aplicación java tiene definido un monitor. Un monitor es una estructura de datos y funciones que únicamente puede ser usado por una hebra a la vez. Cuando otra hebra trata de ejecutar los métodos de un objeto que tiene activo el monitor y está ocupado por otra hebra se bloqueará hasta que la primera termine de utilizarlo. El objeto activará su bloqueo cuando una de las hebras ejecuta uno de sus métodos o segmento de código del tipo synchronized (sin que esto evite que se puedan ejecutar otros métodos del objeto que no estén etiquetados como synchronized).
- Los objetos en Java tienen además una variable de espera, en la que las diferentes hebras de la aplicación pueden esperar cuando ejecutan el método wait() de dicho objeto. Las hebras que esperan en esta variable de espera despertarán cuando otra hebra ejecute el método notify() del propio objeto (despertando en este caso una de las hebras de espera al azar) o el método notifyall(), que despierta a todas las hebras que están en espera.

Ésta será la manera en que haremos que los diferentes elementos y el proceso lógico detendrán su ejecución en espera de ser despertados por la hebra correspondiente.

En lo que respecta a la implementación existían dos posibles soluciones. La primera de ellas pasaba por tener un único objeto en el que las diferentes hebras de la aplicación puedan hacer esperas. Sin embargo, el método notify() no permite seleccionar cual de las hebras detenidas debe ser la que se despierte (no podemos seleccionar la que queremos despertar), así que deberíamos añadir las estructuras necesarias para poder solventar este problema. La segunda de ellas es disponer de un objeto (para usar como monitor y variable de espera) para cada una de las hebras que deben ser detenidas y despertadas durante la ejecución. Ésta fue la manera en que se programaron nuestras librerías. Para ello implementamos una clase de objetos denominado Bloqueo, representado en la figura 6.

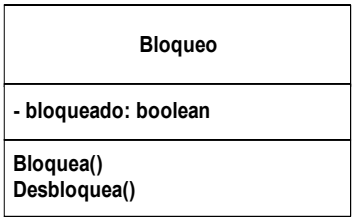

Figura 6: Clase Bloqueo

Cada uno de los objetos que ejecutan como hebras independientes en el sistema tendrán un atributo que será un objeto de la clase Bloqueo, de forma que para deterse deberá hacer un Bloquea() sobre su atributo, y cuando alguna otra hebra quiera despertarlo deberá hacer un Desbloquea() sobre el atributo de la clase Bloqueo del objeto bloqueado. De esta forma, no habrá confusión sobre como despertar a un hilo en concreto. Para darle más funcionalidad implementamos el código de bloqueo como el de un semáforo [2], código se muestra en la figura 7.

```
public class Bloqueo {
  private int conteo;

  public Bloqueo() {
   conteo = 0;
  }

  public synchronized void Bloquea() throws
InterruptedException {
   conteo--;
   while (conteo < 0) wait();
  }

  public synchronized void Desbloquea() {
   conteo++;
   if (conteo == 0) notify();
  }
}
```

Figura 7: Código de la clase bloqueo

4.4. **SEGUNDA APROXIMACIÓN. LA CLASE RECURSO**

La siguiente aproximación a nuestra librería de simulación fue introducir las clases necesarias para implementar los "recursos" que deben usar los elementos del sistema para poder desarrollar su actividad. En un principio nos planteamos los recursos como un tipo de entidades "pasivas", es decir, por si mismos no son capaces de consumir tiempo de simulación. El siguiente paso fue considerar que los recursos sólo estarán disponibles en un determinado horario, con lo que deben ser capaces de hacer esperas en tiempo, para ponerse disponibles en su horario. Y finalmente se abordó el caso en el que un recurso pueda potencialmente realizar varias actividades en un mismo horario, por ejemplo, un médico que está pasando consulta puede ser llamado desde urgencias. Para resolver esta cuestión se implementó la clase 'GestorActividades' que agrupa todas las actividades relacionadas por los mismos recursos. Por lo que es la unidad mínima en la que se puede partir el problema en el momento de la paralelización. La función que deben tener los recursos en el sistema de simulación es la siguiente:

- Los elementos solicitarán recursos para utilizarlos. Si hay unidades disponibles de éstos recursos, entonces los adquirirán. Si por el contrario no hubieran unidades suficientes para atender su petición, deben detener su ejecución hasta que puedan serle asignados los recursos solicitados. Nuevamente, será necesario tener mecanismos para detener la ejecución de un elemento y volverla a reanudar.
- Los recursos, por otra parte, deberán mantener una cola de peticiones que tienen pendientes (no han podido aún ser atendidas), para que cuando el elemento que esté haciendo uso de él finalice su función, pueda ser entregardo a un nuevo elemento que esté a la espera.
- Los recursos entonces deben llevar una gestión de todas las peticiones que tienen pendientes y que aún no han podido atenderse.

Finalmente ha resultado que la librería implementada tiene las clases que se muestran en la figura 8.

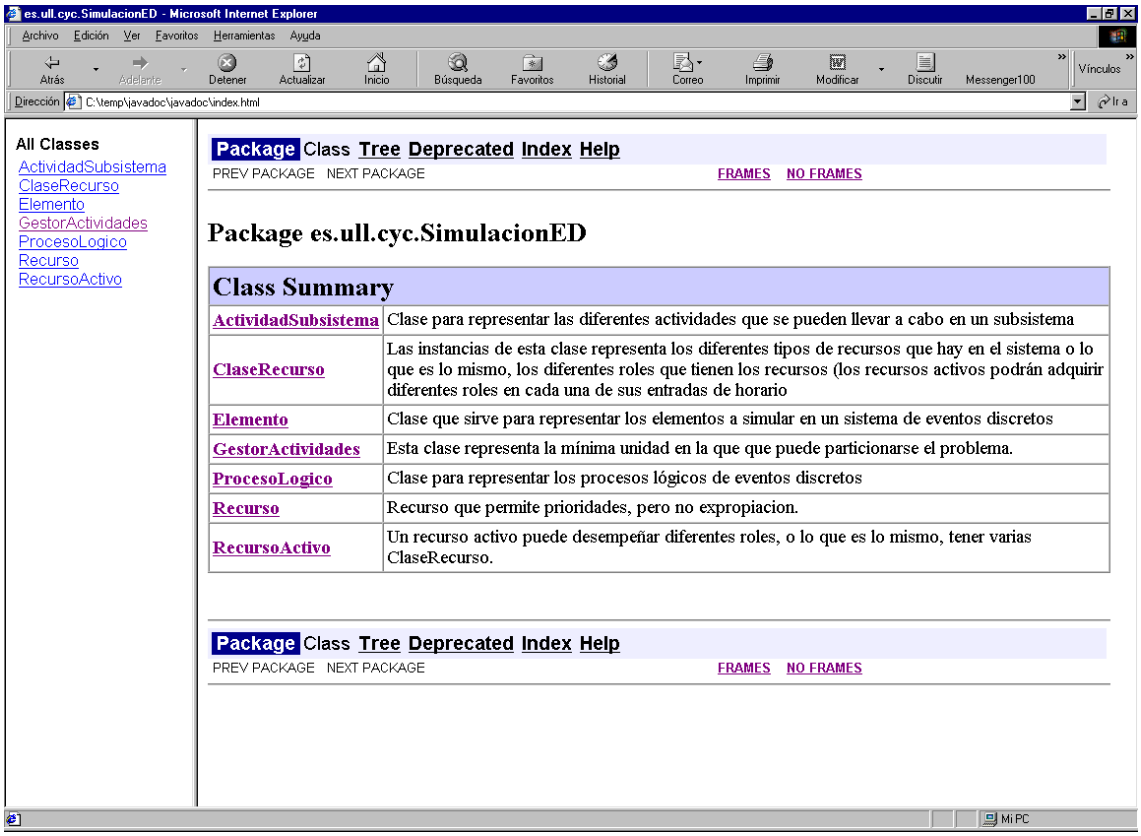

Figura 8: Clases de la librería Java para la simulación de sistemas de eventos discretos.

6. CONCLUSIONES

El presente trabajo se enmarca en una línea de investigación desarrollada por miembros del Grupo de Computadoras y Control de la Universidad de La Laguna para el estudio de la gestión en las organizaciones sanitarias. En este artículo se ha descrito un paquete software para la simulación de sistemas de eventos discretos, que consiste en una librería de funciones genéricas para la realización de este tipo de simulaciones. La librería se ha implementado en Java y se justifica, en este artículo, cuales son las características de este lenguaje de programación que nos han llevado a elegirlo como herramienta de desarrollo. Actualmente se trabaja en la simulación secuencial pero se ha diseñado para poder realizar simulaciones en paralalo.

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.

Referencias

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

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

[3] Keith Lea, The Java is Faster than C++ and C++ Sucks Unbiased Benchmark, http://kano.net/javabench/

[4] L.Moreno, R.M.Aguilar, C.A.Martín, J.D. Piñeiro, J.I.Estévez, J.F.Sigut, J.L.Sánchez, (2000) “Simulation”, *Patient-Centered Simulation to Aid Decision-Making in Hospital Management* , Vol. 74/5, pp. 290 - 303.

[5] Michael J. Radwin, http://www.radwin.org/michael/

[6] Michael Pidd, (1998), Computer Simulation in Management Science, Ed. Wiley