Apuntes de Simulación Informática

Ingeniería Informática -II56-

Autor: Rafael Berlanga Llavori

Curso 2005–2006

Capítulo 1

Sistemas y Modelos de Simulación

1.1. Introducción

La Simulación es una herramienta de análisis que nos permite sacar conclusiones sin necesidad de trabajar directamente con el *sistema real* que se está simulando. La simulación es especialmente útil cuando no disponemos de dicho sistema real o nos resulta demasiado arriesgado realizar experimentos sobre él.

Posiblemente muchos de vosotros asociéis la simulación con los famosos simuladores de vuelo con el que habréis jugado alguna vez. En general, se podría considerar que muchos juegos de ordenador son simuladores, ya que recrean parte de la realidad. Sin embargo, su intención no es el análisis sino el ocio, y esto es una diferencia bastante importante.

En muchas áreas de ingeniería se utilizan los simuladores como una herramienta de trabajo más. Por ejemplo, en el diseño de nuevos fármacos se suelen utilizar modelos moleculares que sirven para simular mediante un ordenador la interacción entre compuestos químicos. Los ingenieros de automóviles también utilizan modelos computerizados para analizar el impacto de los choques en la seguridad de los viajeros. Seguramente a ti también se te ocurran otros escenarios donde se utiliza un simulador como herramienta de trabajo.

Es importante observar que, al contrario que en los juegos que hemos mencionado al principio, en la simulación existe un motivo especial (objetivo) por el cual se lleva a cabo la simulación. Además, en toda simulación es necesario realizar muchas pruebas para llegar a unas conclusiones, es decir requiere un proceso de experimentación.

Otro tipo de simulación de la que posiblemente habéis oído hablar es la Simulación Numérica. Muchos métodos de integración se basan en este tipo de simulación. Básicamente, estos métodos se basan en la generación de números aleatorios para la obtención de una solución aproximada. Un buen ejemplo de este tipo de simulación es el método de Monte-Carlo, para calcular integrales de forma aproximada. Sin embargo, en la simulación numérica no se simula ningún sistema real, ni existe un proceso de experimentación para sacar conclusiones.

Desde el punto de vista de la Ingeniería Informática lo que nos interesa es simular los sistemas informáticos con los que trabajamos habitualmente, por ejemplo,

sistemas distribuidos, redes de ordenadores, bases de datos, entornos de multiprogramación, etc. En general, nos interesará simular el comportamiento de aquellos Sistemas Informáticos que bien por su complejidad o por su coste elevado no podemos estudiar ni mediante técnicas analíticas (modelos matemáticos directamente resolubles) ni a través de pruebas reales. Así pues, la simulación resultará especialmente útil en la fase de estudio de viabilidad, previa al desarrollo e implantación de un Sistema Informático.

1.1.1. Ciclo de desarrollo de un Simulador

Antes de comenzar a diseñar un simulador debemos tener bien claro cuál es el *objetivo* de la simulación. En un sistema informático, los objetivos de la simulación suelen ser dos: estimar el tiempo medio de respuesta del sistema, y estimar la capacidad máxima del sistema (número máximo de peticiones por unidad de tiempo que puede soportar el sistema).

Ejemplos de objetivos son: estimar el retraso medio de los paquetes en una determinada topología de red, estimar el tiempo medio de respuesta de una Base de Datos distribuida para un perfil de consulta determinado, estimar el número máximo de clientes simultáneos que puede soportar un servidor, etc.

En la figura 1.1 se muestra el ciclo completo de desarrollo de un simulador, el cual pasamos a describir a continuación.

Modelo del Sistema. Una vez fijados los objetivos, debemos analizar la estructura y comportamiento del sistema e intentar plasmarlo en papel. Es decir, debemos diseñar un *modelo* que abstraiga las partes más relevantes del sistema. En esta parte del diseño es muy importante seleccionar bien qué partes del sistema son interesantes modelar, siempre sin perder de vista los objetivos de la simulación. Por ejemplo, en una red de estaciones donde queremos medir el retraso medio de los paquetes sólo debemos modelar las partes del sistema que afectan a dicho retraso, es decir: las colas de procesos que se forman en las estaciones y el tiempo medio de transmisión de un paquete. En las siguientes secciones mostraremos las distintas técnicas de modelado para sistemas que utilizaremos durante el curso.

Modelo Computacional. Cuando el modelo ya ha sido suficientemente especificado (y a ser posible validado por algún experto), pasaremos a la implementación del simulador. Para ello utilizaremos programas, o paquetes de programas, que contengan las rutinas necesarias para realizar nuestro simulador. Entre otras cosas, estos paquetes o programas deben contener como mínimo las siguientes funciones:

- Generadores de números y variables aleatorias.
- Gestor de eventos y/o procesos.
- Funciones de análisis estadístico.

Experimentación con el simulador. Una vez implementado y depurado el simulador, debe diseñarse la batería de pruebas que nos permita extraer los resultados, y a partir de éstos las conclusiones de la simulación. Esta última etapa es la más

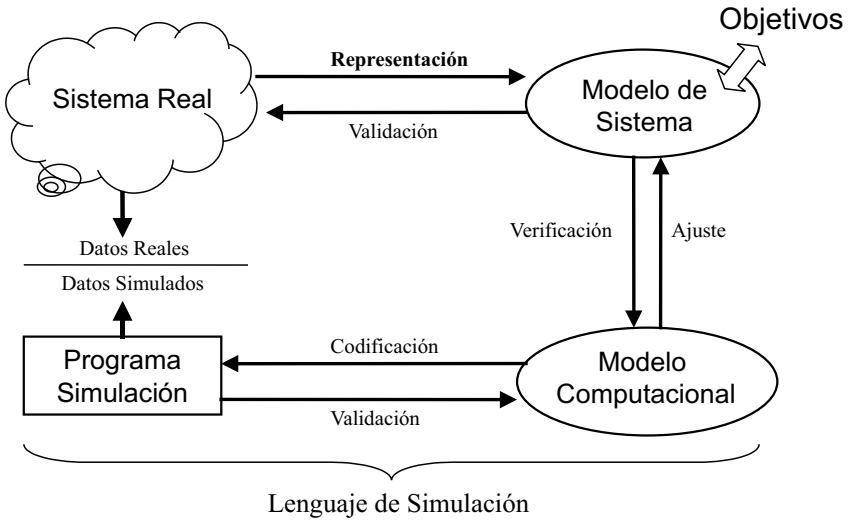


Figura 1.1: Ciclo de desarrollo en Simulación.

importante y delicada de todo el proceso de simulación. Cualquier error en el modelo o su implementación desembocará en un error importante en las conclusiones. Asimismo, un error en la selección de la batería de pruebas también puede conducir a un error en las conclusiones finales. Por esta razón, antes de experimentar con el simulador es muy importante asegurarse de que refleja fielmente el modelo, y que éste no contiene incongruencias con respecto al sistema real. Es decir debe ser *correcto* y *válido*.

1.2. Herramientas para el modelado de sistemas

En esta sección vamos a revisar las principales herramientas de modelado que utilizaremos en este curso para construir los modelos de nuestros simuladores. Antes de entrar en materia, es importante resaltar los aspectos del sistema real que vamos a tener en cuenta durante su modelado:

- *La estructura del sistema*: las partes de las que está compuesto el sistema, y cómo se relacionan entre sí.
- *La dinámica del sistema*: cómo evoluciona el sistema en el tiempo, los cambios que en él acontecen.
- *Los recursos del sistema*: qué partes del sistema son compartidas por distintos agentes y cómo se gestiona su servicio.

En las siguientes secciones vamos a ver distintas herramientas para modelar estos aspectos del sistema real.

1.2.1. Estructura de un sistema

El elemento más básico en un modelo de simulación es la *variable de estado*. Una variable de estado representa una parte del sistema que cambia con el tiempo. Ejemplos de variables de estado son el tamaño de una cola de espera, el número de clientes procesados en un servidor, etc.

Cuando se modela sistemas muy complejos, resulta bastante complicado identificar todas las variables de estado que nos van a interesar para la simulación. Para estos casos, resulta más adecuado utilizar alguna metodología de modelado que sea capaz de representar entidades complejas y sus relaciones. En nuestro caso, la metodología más apropiada y familiar es la Orientación a Objetos, debido principalmente a los siguientes motivos:

- Permiten varios grados de *abstracción* del sistema. Utilizando la modularidad y aplicando refinamientos sucesivos podemos ir representando los distintos niveles de detalle del sistema.
- Podemos ocultar la estructura y ciertos detalles de los objetos (incluida su implementación final).
- Podemos construir nuevos modelos re-utilizando el conocimiento adquirido en otros anteriores.
- *Escalabilidad*, podemos aumentar el modelo añadiendo nuevos módulos sin que ello suponga un mayor esfuerzo.

En este curso, utilizaremos la notación gráfica del modelo lógico de UML (Unified Modelling Language). UML es el lenguaje de modelado más aceptado en el diseño de Sistemas de Información. Soporta una notación gráfica para describir los distintos elementos empleados en Ingeniería del Software: interfaces, clases, objetos, estados, componentes, flujo de datos, etc. En concreto, el modelo lógico de UML constituye la vista estática de los objetos y las clases del modelo.

Clases y Objetos

En cursos anteriores seguramente ya te habrás familiarizado con la terminología de la Orientación a Objetos (OO), sobre todo a nivel de programación. Muchos de los conceptos que ya conoces en programación OO son utilizados de forma directa en el modelo lógico de UML. Por esta razón no vamos a profundizar demasiado en los conceptos básicos del modelo Orientado a Objetos.

Un *clase* es una especificación abstracta de un conjunto de objetos que comparten una serie de *atributos* y un *comportamiento*. Desde el punto de vista de la Simulación, los atributos de los objetos van a representar las *variables de estado* de ese objeto. El comportamiento de una clase se define con una serie de métodos, los cuales permiten manipular el estado de los objetos sin necesidad de conocer su estructura interna. Esta propiedad se conoce como *encapsulación*.

Para describir los métodos utilizaremos la siguiente notación:

$$nombre\_metodo(Clase_1, \ldots, Clase_k) : Clase$$

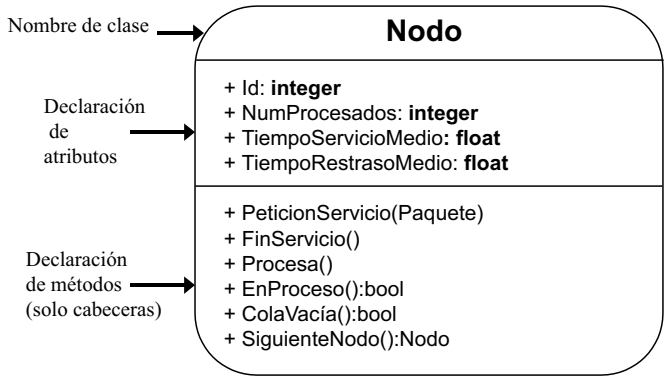


Figura 1.2: Ejemplo de clase en UML.

Fíjate que junto con el nombre del método se especifican los objetos que sirven de entrada y el objeto resultado de su aplicación (todos son opcionales).

En el modelo lógico de UML, una clase se representa como una caja con tres secciones: el nombre de la clase, los atributos de la clase y el comportamiento de la clase (métodos). En la figura 1.2 se muestra un ejemplo de clase en UML, para representar los nodos de una red de ordenadores.

En UML los atributos y métodos se pueden marcar como *públicos* (+), *privados* (-) y *protegidos* (#), para indicar los distintos grados de visibilidad de éstos con respecto al que usa la clase. En el primer caso, se indica que los atributos son visibles para todos, en el segundo caso que no son visibles para las clases externas, y en el último caso se indica que los atributos solo son visibles para las subclases de la clase.

En UML existen otras marcas sobre métodos y atributos, pero no resultan de interés en los modelos de Simulación.

Un *objeto* en UML es simplemente una caja con el nombre de la clase del objeto y su identificador único. A partir de ahora, utilizaremos la siguiente notación: $O.at$ representa el atributo at del objeto O, y $O.m()$ representa el método $m()$ del objeto O. Fíjate que ésta es una notación similar a la utilizada en C++ o en Java.

Relaciones

Las relaciones entre las clases representan la estructura del sistema en forma abstracta. Los tipos de relaciones que tendremos en cuenta en nuestros modelos son las siguientes:

- **Agregación :** representa relaciones de composición (parte-de) entre objetos del modelo. Con esta relación expresaremos por ejemplo que un objeto "red" se compone de objetos "nodo", o que un "servidor" se compone de 3 objetos "discos" y un objeto "CPU". En el diagrama UML podemos anotar la cardinalidad de la relación en los extremos de la línea que representa la relación (ver figura 1.3).

- **Asociación :** representa cualquier tipo de relación entre entidades del modelo. Cada objeto que forma parte de la relación puede tomar un *rol*. Por ejemplo,

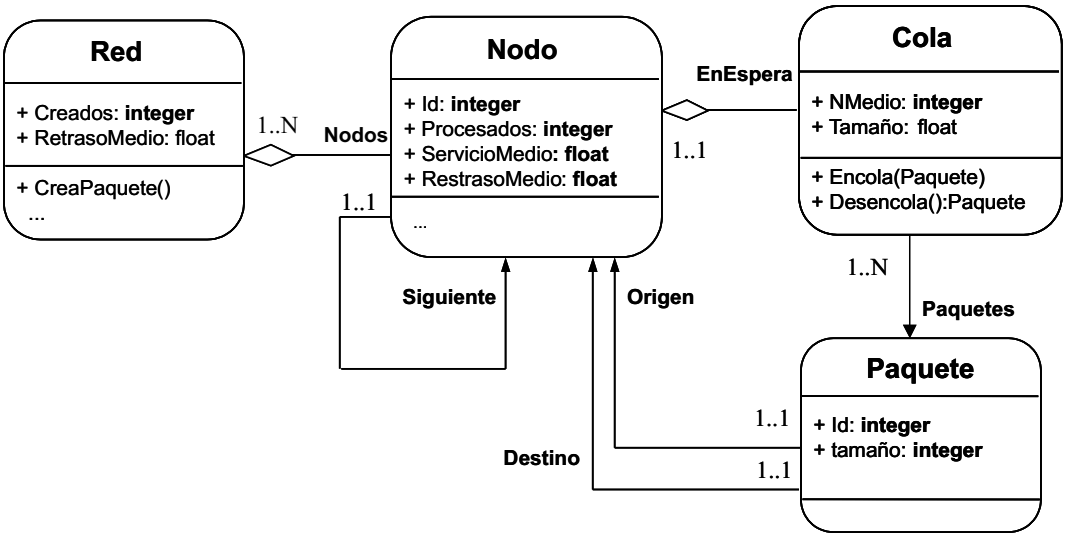


Figura 1.3: Ejemplo de diagrama de clases un UML.

el objeto "navegador" se relaciona con el objeto "servidor Web", el primero toma el rol de "cliente" y el segundo toma el rol de "servidor". Al igual que en la agregación, podemos indicar la cardinalidad de la relación.

- **Herencia :** representa relaciones de herencia entre clases. Una subclase hereda la estructura y comportamiento de su/s superclase/s. En UML la subclase siempre apunta a la superclase.

En la figura 1.3 se muestra un diagrama completo de clases donde se pueden ver varios ejemplos de las relaciones anteriores. El diagrama corresponde al modelo de una red de ordenadores.

Para más información sobre UML, en la dirección http://www.dsic.upv.es/ũml disponéis de un curso completo, así como diversos enlaces a recursos y material sobre el tema.

1.2.2. Diagramas de Sucesos Discretos

Con los diagramas UML podemos reflejar fácilmente las entidades del sistema con sus variables de estado y sus relaciones. En esta sección veremos cómo expresar la evolución de sistema a lo largo del tiempo. Para ello, introduciremos una notación gráfica para representar los sucesos de un sistema y cómo se relacionan éstos a lo largo del tiempo.

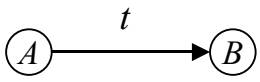
Un *suceso* representa un cambio en alguna variable de estado del modelo. Por simplicidad, asumiremos que estos cambios son instantáneos. Por ejemplo, consideremos los siguientes cambios en el modelo del ejemplo de la figura 1.3:

- *Creación de un paquete*, el cual incrementa la variable de estado $Red.Creados$.
- *Llegada de un paquete a un nodo* N, que incrementa la variable $N.Recibidos$, y modifica la cola de espera $N.Cola$.

- *Procesamiento de un paquete en un nodo N*, que incrementa la variable $N.Procesados$ y modifica la cola de espera $N.Cola$.

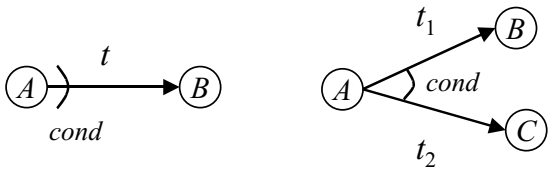
Cada uno de estos cambios representa un *suceso* aislado del sistema. Lo siguiente que tenemos que hacer es relacionar estos sucesos en el tiempo para expresar la dinámica del sistema.

La primera relación básica entre sucesos es la de *planificación* o relación *causa-efecto*. Con esta relación expresamos que después del suceso A va a ocurrir siempre el suceso B después de t unidades de tiempo ($t \geq 0$). Gráficamente lo representaríamos como:



Por ejemplo, la creación de un paquete causa necesariamente su llegada a un nodo de la red, y el tiempo transcurrido entre ambos será el tiempo de transmisión por la red. Como veremos más adelante, el tiempo entre sucesos será muchas veces una variable aleatoria que expresa la variabilidad de éstos.

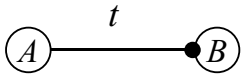
La segunda relación básica es la *planificación condicional*. Con esta relación expresamos que un suceso A causará la ocurrencia de un suceso B en t unidades de tiempo si se cumple una condición *cond* sobre las variables del sistema. Esta relación se representa como sigue:



Un ejemplo de esta relación sería la planificación del suceso de procesamiento de un paquete en un nodo si se cumple que la cola de espera del nodo está vacía.

La condición de planificación *cond* puede contener variables aleatorias para representar causas que se dan con una determinada probabilidad. Por ejemplo, un paquete procesado en un nodo podría tener distintas probabilidades de ir a los distintos nodos de la red.

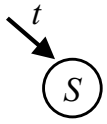
Otra relación importante entre sucesos es la *cancelación de sucesos*, la cual expresa que un suceso A provoca la cancelación de un suceso ya planificado B, transcurridas t unidades de tiempo. Esta relación también puede ser condicional, así como contener variables aleatorias. Su representación es como se muestra a continuación:



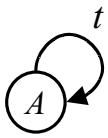
Un ejemplo de cancelación es la interrupción de un proceso debido a un fallo en el nodo, o cuando el tiempo estimado para terminar un proceso cambia debido a un cambio en la carga del nodo.

Además de las relaciones anteriores, existen otros elementos en el diagrama de sucesos que resultan necesarios para la simulación del sistema, a saber:

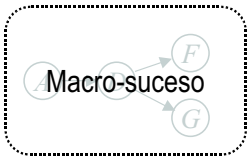
- **Sucesos iniciales:** los cuales representan por un lado las condiciones iniciales de la simulación, y por otro lado indican los puntos de entrada del diagrama de sucesos. Los sucesos iniciales los representaremos colocándoles una flecha de entrada.



- **Sucesos periódicos:** los cuales representan la repetición periódica de un suceso en t unidades de tiempo. Con estos sucesos se modela la llegada de clientes a un sistema, las roturas periódicas de una máquina, etc. Estos sucesos los representaremos como sigue:



- **Macro-sucesos:** se representa en forma de caja, y permite ocultar los sucesos de una parte del modelo. La idea es que a la hora de diseñar un diagrama primero se especifiquen los macro-sucesos y después se vayan refinando éstos por separado. Por ejemplo, todos los sucesos internos que se producen en un nodo podrían representarse con el macro-suceso "Proceso-en-Nodo", ocultando los detalles del mismo.



Ejercicios. Describe mediante diagramas de suceso los siguientes sistemas:

1. Un semáforo con los estados rojo, amarillo y verde.

2. Una máquina expendedora de bebidas. El usuario primero introducirá una cantidad de dinero y después realizará la petición deseada (café, cortado, etc.) Si la cantidad introducida no es suficiente, la máquina solicitará al usuario que introduzca más dinero. En caso contrario, la máquina pasará a preparar la bebida solicitada, cuyo tiempo de preparación dependerá del tipo de bebida. Finalmente, si la cantidad introducida era mayor que el coste de la bebida, la máquina devolverá el cambio correspondiente.

3. Un sistema de cuatro procesadores que distribuye uniformemente las tareas que recibe, según la carga de los procesadores.

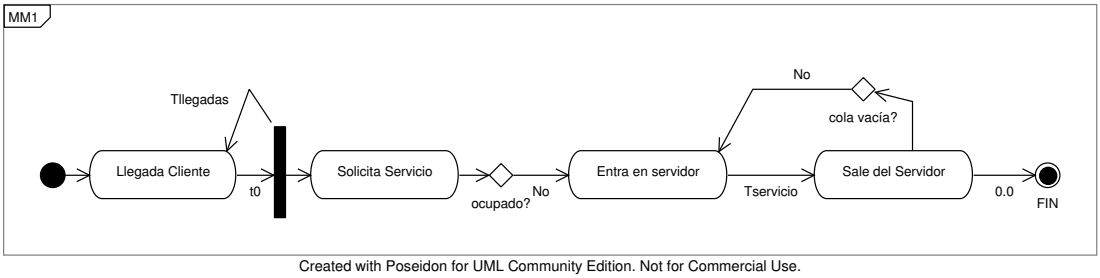


Figura 1.4: Diagrama UML de actividades para una estación de servicio.

4. Un sistema cliente-servidor donde un cliente envia una petición de forma simultánea a cinco servidores.

Aunque UML no soporta directamente la especificación de diagramas de sucesos temporizados como los que hemos visto en esta sección, si que es posible expresar los mismos mediante los denominados *diagramas de actividades*. En la figura 1.4 se muestra un diagrama de este estilo editado con la herramienta Poseidon [1], y puede compararse con el de la figura 1.6 que representa el diagrama de sucesos equivalente.

1.2.3. Recursos del sistema

Un *recurso* es una entidad del sistema que debe ser compartida por varios agentes. En los sistemas informáticos resulta frecuente encontrar recursos compartidos (ej.procesadores, servidores de bases de datos, nodos de una red, etc.), de ahí que sea crucial poder modelarlos de forma correcta.

El modelo más aceptado para representar los recursos de un sistema son las denominadas *redes de colas de espera*. En esta representación, las entidades básicas son las denominadas *estaciones de servicio*, las cuales están conectadas entre sí para expresar el flujo de las peticiones de los agentes (también denominados *clientes*) en los distintos recursos del sistema.

Sobre este modelo se define una teoría estadística, denominada *teoría de colas*, que es utilizada para evaluar analíticamente la eficiencia y fiabilidad de sistemas con recursos. Sin embargo, la aplicación de esta teoría resulta limitada debido a la complejidad de los sistemas reales, y en estos casos se recurre a las técnicas de simulación.

En general, una estación de servicio se caracteriza por los siguientes aspectos:

- La tasa de llegada de clientes a la estación.
- El tiempo que tarda en servirse cada cliente en la estación.
- El número de servidores de la estación.

[1] http://www.gentleware.com/index.php?id=ce

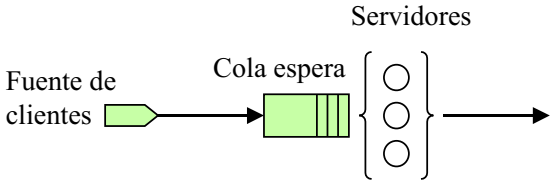


Figura 1.5: Notación gráfica para una estación de servicio.

- La capacidad del sistema, es decir, el número máximo de clientes que pueden esperar a ser servidos.
- La política de servicio, es decir, cómo se va a organizar el procesamiento de los clientes que esperan en la cola.

En la figura 1.5 se muestra la notación gráfica que se utiliza para representar una estación de servicio.

La siguiente tabla muestra el conjunto de variables de estado que se asocia a una estación de servicio de m servidores:

T	Tiempo entre llegadas (variable aleatoria)
$\lambda = 1/E[T]$	Tasa media de llegada (E es el valor esperado de T)
S	Tiempo de servicio de cada trabajo (variable aleatoria)
$\mu = 1/E[S]$	Tasa media de servicio
$I = \frac{\lambda}{\mu}$	Intensidad de tráfico (con $I > m$ el sistema está saturado)
N	Número de trabajos en la estación (variable aleatoria discreta)
N_q	Número de trabajos en espera (variable aleatoria discreta)
N_s	Número de trabajos recibiendo servicio (v.a. discreta)
R	Tiempo de respuesta del sistema (variable aleatoria)
W	Tiempo de espera en cola (variable aleatoria)

En la teoría de colas, una estación de servicio se especifica con la siguiente notación:

Distribución_Llegada/Distribución_Servicio/Número_Servidores

donde cada distribución puede identificarse con uno de los siguientes términos:

M	distribución exponencial (proceso de poisson)
E_k	distribución de una k-Erlang (suma de k procesos de poisson)
D	distribución determinística (tiempo constante)
H_k	distribución hiperexponencial con parámetro k
G	distribución general

Por ejemplo, el modelo M/M/1 representa de una estación de servicio de un servidor, con un proceso de llegada de poisson (exponencial) y un tiempo de servicio exponencial.

Si una estación de servicio es estable ($I \leq m$), entonces se cumples las siguientes fórmulas (fórmulas de Little):

$$\bar{N}_q = \lambda \bar{W}$$

$$\bar{N} = \lambda \bar{R}$$

$$\bar{N} = \bar{N}_q + I$$

Es decir, la media de clientes en la cola y la media de clientes en la estación son directamente proporcionales a las medias de los tiempos de espera y de los tiempos de respuesta, respectivamente. La relación entre ellas es la propia tasa de llegada.

Políticas de servicio

En cuanto a la *política de servicio* de una estación, podemos seleccionar una de las siguientes:

- *Firts Come First Served* (FCFS o FIFO): los clientes se sirven en orden de llegada. Esta política tiene como propiedad que todos los clientes esperan el mismo tiempo de media, independientemente del tiempo de servicio que empleen. Es decir, tanto las tareas largas como las cortas esperan de media lo mismo.

- *Last Come First Served* (LCFS o LIFO): los clientes se sirven en orden inverso al de llegada. Cuando llega un cliente nuevo, este pasa a servirse de inmediato (con apropiación), encolando al cliente que se estaba sirviendo en su caso. Esta política castiga a las tareas más largas, que deben interrumpirse cada vez que llega una nueva tarea. Si la tarea es lo suficientemente corta, no sufrirá interrupciones.

- *Round-Robin* (RR): todos los clientes van sirviéndose en incrementos de tiempo δt hasta completar sus respectivos tiempos de servicio. Al igual que la política anterior, ésta penaliza las tareas largas. De hecho, analíticamente se comprueba que las tareas con un tiempo inferior a $\bar{S^2}/(2 \cdot \bar{S})$ esperan menos tiempo que en el caso de una FCFS. Por contra, todas las que superen dicho tiempo esperarán un tiempo proporcional a su tiempo de servicio, que siempre será mayor que si se procesasen según la FCFS. Esto puede provocar que el sistema se sature con facilidad, dependiendo de la varianza de los tiempos de servicio ($\bar{S^2}$).

- *Processor Sharing* (PS): todos los clientes se sirven simultáneamente, pero con tiempos de servicio proporcionales al número de clientes que se están procesando en ese momento. Esta política es un caso particular de la anterior, donde los incrementos de tiempo δt son prácticamente cero.

- *Service in Random Order* (SIRO): se toma aleatoriamente el siguiente cliente a procesar.

- Con control de *prioridades*, las cuales pueden ser:

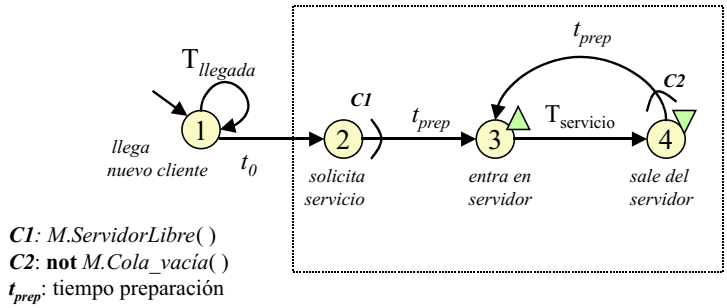


Figura 1.6: Diagrama de sucesos para una estación FCFS

- Expulsiva con reanudación (*preempt*): el cliente con más prioridad interrumpe al que se está sirviendo en ese momento, y el cliente interrumpido se encola y termina lo que le falta.
- Expulsiva con reinicialización: el cliente con más prioridad interrumpe al que se está sirviendo en ese momento, y el cliente interrumpido se encola y empieza de nuevo su servicio.
- No expulsiva: el cliente con más prioridad espera al que se está sirviendo.
- Dinámica: donde la prioridad de un cliente depende del número de veces que ha entrado en la estación.

En la figura 1.6 se muestra el diagrama de sucesos para la política de servicios más común, que es la FCFS, y dejamos como ejercicio realizar los diagramas de sucesos para el resto de políticas de servicio. En esta figura, el suceso ① representa la llegada de clientes, los cuales llegan cada $T_{llegada}$ unidades de tiempo. Cada uno de estos clientes planifica una petición a la estación de servicio (suceso ②). Si existe algún servidor libre, se planifica el procesamiento de la petición en t_{prep} unidades de tiempo. En el suceso ③ comienza el procesamiento de la petición, y en el suceso ④ termina. Por otro lado, hemos incluido dos símbolos en el diagrama de sucesos para representar la toma (Δ) y la liberación (∇) de una estación de servicio por parte de un cliente. Finalmente, cada vez que termina una petición, si la cola de espera no está vacía, se pasa a procesar la siguiente petición.

Ejecución de un diagrama de sucesos. Para ilustrar el funcionamiento de la estación descrita en la figura 1.6, supondremos que sus parámetros son los siguientes: $T_{llegada} = 0{,}4$, $T_{servicio} = 0{,}5$ y $t_{prep} = 0$. A continuación mostramos un fragmento de la traza de ejecución de la estación de servicio. En la primera columna se muestra el tiempo de simulación, en la segunda el suceso que se está ejecutando junto con el cliente implicado, las siguientes dos columnas son las variables de estado de la estación (servidor libre y la cola de peticiones). Finalmente, la última columna representa los sucesos planificados tras la ejecución de cada suceso (el que está marcado con $\surd$ es el siguiente suceso que va a ser ejecutado). En la tabla hemos sombreado los cambios producidos por el suceso en ejecución.

T_{sim}	*Ejecuta* (Suceso/Cliente)	*M.Libre*	*M.Cola*	*Planifica* (Suceso/Cliente, Tiempo)
0.0	-	libre	[ ]	(1/c1, 0.0) √
0.0	1/c1	libre	[ ]	(1/c2, 0.4) (2/c1, 0.0) √
0.0	2/c1	libre	[ ]	(1/c2, 0.4) (3/c1, 0.0) √
0.0	3/c1	ocupado	[ ]	(1/c2, 0.4) √ (4/c1, 0.5)
0.4	1/c2	ocupado	[ ]	(1/c3,0.8) (2/c2,0.4)√ (4/c1,0.5)
0.4	2/c2	ocupado	[c2]	(1/c3,0.8) (4/c1,0.5) √
0.5	4/c1	ocupado	[c2]	(1/c3,0.8) (3/c2,0.5) √
0.5	3/c2	ocupado	[ ]	(1/c3,0.8)√ (4/c2,1.0)
...	...	...	...	...

1.2.4. Cambios contínuos

Hasta ahora nos hemos centrado principalmente a representar sistemas informáticos mediante sucesos discretos y modelado de recursos. Sin embargo, no todos los sistemas pueden ser representados con estas herramientas. En particular, los modelos físicos asumen un tiempo continuo sobre el cual las variables de estado evolucionan también de forma continua. En este contexto los cambios no pueden modelarse como transiciones instantáneas de estado, sino que es necesario modelar los cambios infinitesimales que se van produciendo sobre las variables de estado. A este tipo de simulación se denomina *Simulación Continua*.

Los modelos de simulación continua se representan con un conjunto de variables de estado continuas (x, y, etc.) y las ecuaciones diferenciales que definen sus cambios. Una ecuación diferencial tiene la siguiente forma:

$$b_n \frac{d^n x}{d^n t} + b_{n-1} \frac{d^{n-1} x}{d^{n-1} t} + \ldots + b_1 \frac{dx}{dt} + b_0 = 0$$

En general, las ecuaciones diferenciales de un sistema físico no suele pasar del segundo orden, y por esta razón se utiliza la siguiente notación para expresar las derivadas parciales de primer y segundo orden:

$$\ddot{x} = \frac{d^2 x}{d^2 t}$$

$$\dot{x} = \frac{dx}{dt}$$

En la figura 1.7 se muestra un ejemplo sencillo de modelo de un sistema físico. No vamos a profundizar mas en el tema de Simulación Continua debido principalmente a que no es necesaria para modelar nuestros Sistemas Informáticos.

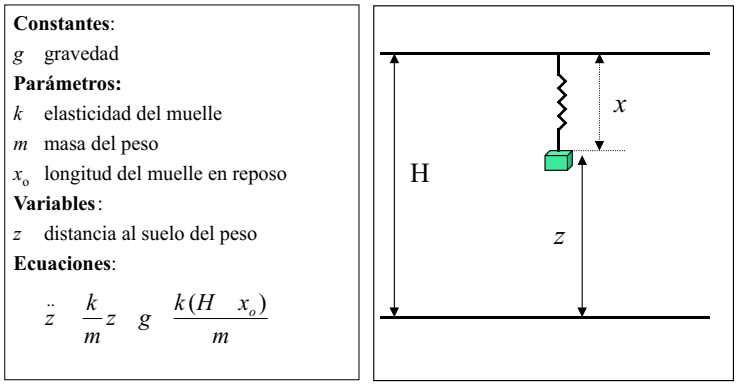


Figura 1.7: Ejemplo de modelo de Simulación Continua

1.3. Ejemplos de modelos

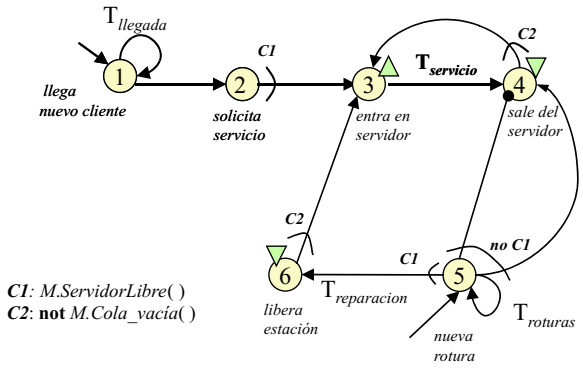
En esta sección vamos a desarrollar dos modelos de ejemplo. El primero es un modelo simple de un sistema genérico con una estación de servicio que sufre interrupciones debido a la presencia de roturas. El segundo ejemplo es un modelo de un sistema más complejo donde se combinan varias estaciones de servicio.

1.3.1. Estación con roturas

Los sistemas informáticos suelen presentar fallos en el funcionamiento de sus componentes que requieren ser modelados para aproximar con mayor exactitud su respuesta real. Ejemplos de fallos en un sistema pueden ser: las roturas físicas de algunas partes, lo que supone poner fuera de servicio esas partes durante mucho tiempo, paradas temporales debidas a ajustes o manteniemiento de alguna componente del sistema, interrupciones temporales en las conexiones de red, etc.

La forma de modelar las roturas suele realizarse extendiendo el modelo de la estación FCFS (o con cualquier otro tipo de política de servicio) con un nuevo suceso periódico (suceso ⑤) que replanifica el suceso de final de servicio (suceso ④) de la figura 1.6). Es decir, cada cierto tiempo (periodo de la rotura) se interrumpe el servicio de la estación, y se replanifica el servicio para dentro de $T_{reparacion} + T'_{servicio}$ unidades de tiempo, donde $T_{reparacion}$ es el tiempo en el que se tarda en reparar la estación, y $T'_{servicio}$ es el nuevo tiempo de servicio del cliente interrumpido. Fíjate todo esto debe realizarse si la estación estaba sirviendo a algún cliente (condición *no c*1). Finalmente, también debe tenerse en cuenta el caso en el que la rotura se produce con la estación vacía. En este caso, se bloquea la estación para que no procese a ningún cliente y se planifica su liberación con el tiempo de reparación (suceso ⑥). Como durante la reparación puede haberse encolado algún cliente, debemos planificar el suceso ③ para pasarlo a servir.

De este modo, el diagrama de sucesos correspondiente a una estación con roturas sería el siguiente:



Observa que la *replanificación* de un suceso se realiza en dos pasos: primero se cancela el suceso y después se replanifica con el nuevo tiempo.

El diagrama anterior puede presentar distintas variantes según sea la política de servicio de la estación. Así, si el cliente interrumpido debe expulsarse, entonces se cancela el suceso ④ y se planifica el suceso ③ con tiempo $T_{reparacion}$ para servir al siguiente cliente de la cola. Si en cambio el cliente interrumpido debe volverse a encolar, entonces planificaríamos el suceso ② con ese cliente.

1.3.2. Un modelo complejo

Supongamos un sistema informático compuesto de cuatro terminales (T1..T4), un procesador (CPU) y dos discos (D1 y D2). Los usuarios lanzan transacciones desde los terminales, donde cada transacción consiste en una serie de visitas a la CPU y a los discos. Hasta que una transacción no ha finalizado, no se lanza una nueva. De este modo, como mucho sólo pueden ejecutarse cuatro transacciones a la vez. Veamos un ejemplo de ejecución:

```
Transacción 1: T1 -> CPU -> D1 -> CPU -> D2 -> CPU -> T1
Transacción 2: T2 -> CPU -> D1 -> CPU -> T2
Transacción 3: T3 -> CPU -> D2 -> CPU -> D2 -> CPU -> D1 -> T3
Transacción 4: T4 -> CPU -> D1 -> CPU -> D1 -> CPU -> T4
```

La primera transacción consiste en un acceso a la CPU, seguido de un acceso al disco D1, otro acceso a la CPU, un acceso al disco D2, otro a la CPU, y finalmente se termina la transacción volviendo a la terminal de origen. Todos estos accesos se realizan de forma secuencial (uno detrás de otro). En cambio, las cuatro transacciones se ejecutan simultáneamente.

Las características de los dispositivos del sistema se resumen a continuación:

- El tiempo medio de servicio en la CPU es de 4ms.
- El tiempo medio de servicio en ambos discos es de 14ms.
- La comunicación entre dispositivos es menor de 1ms.
- La probabilidad de acceso a los discos desde la CPU es la misma.

Por otro lado, estamos interesados en estudiar el impacto que tiene el número medio de visitas a la CPU ($Niter$) en las prestaciones del sistema. Por ejemplo, podemos definir algunos objetivos interesantes con este parámetro:

- Estimar la duración media de las transacciones con $Niter = 3$. Para ello, definimos la variable de estado Ttrans.
- Estimar el número medio de transacciones procesadas para un periodo de tiempo determinado (por ejemplo 100s) y con $Niter = 3$. Para ello, definimos la variable de estado Ntrans
- Mostrar la evolución de las medidas anteriores conforme aumentamos $Niter$.

Modelando los recursos. Volvamos a los ejemplos de las transacciones anteriores. Podemos ver que las cuatro transacciones realizan accesos simultáneos tanto a la CPU como a los dos discos. Sin embargo, estos dispositivos sólo pueden atender una petición a la vez. Entonces ¿qué se hace con el resto de peticiones? ¿en qué orden se procesarán?

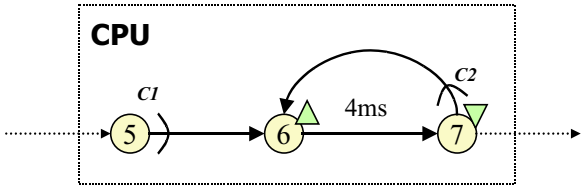
Estos tres dispositivos (CPU, D1 y D2) son los recursos del sistema. En general diremos que un objeto del sistema es un recurso si éste debe ser compartido por varios procesos. Como hemos visto anteriormente, los recursos se modelan como estaciones de servicio, donde hay que identificar una serie de aspectos, entre ellos: la política de servicio, el número de servidores, el tiempo de servicio, la tasa de llegada, su población, etc.

Dado que los tres dispositivos van a procesar las peticiones de una en una, y según su orden de llegada, supondremos que su política de servicio es la FCFS. Así pues, la CPU será una estación de servicio FCFS con un servidor y tiempo de servicio medio de 4ms, y cada uno de los discos será una estación de servicio FCFS con un servidor y tiempo de servicio medio de 14ms.

Observa que en este sistema las tasas de llegada de las estaciones de servicio vienen determinadas por el comportamiento global del sistema, concretamente por el número de peticiones y la longitud de las transacciones.

Diagrama de sucesos. Una vez hemos identificado los recursos, y modelado éstos como estaciones de servicio, tenemos que describir su comportamiento mediante un diagrama de sucesos.

Veamos en primer lugar el diagrama para la CPU:

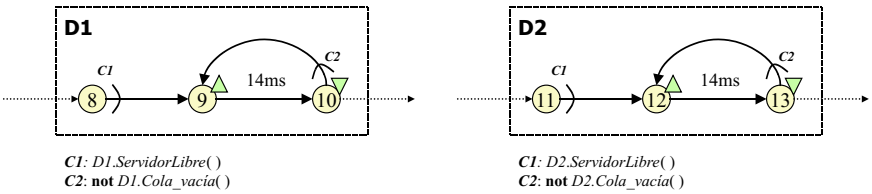


C1: *CPU.ServidorLibre*( )
C2: **not** *CPU.Cola_vacía*( )

Según este diagrama, el procesamiento en la CPU comienza en el suceso ⑤ (los sucesos del ① al ④ los reservamos para los sucesos iniciales). Si la CPU está ocupada,

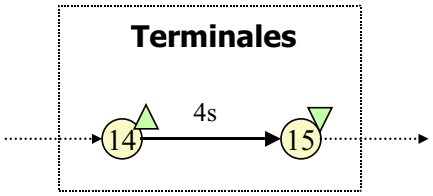
la petición pasará a una cola de espera. En caso contrario, el suceso ⑥ inicia el procesamiento de la petición, y el suceso ⑦ lo termina. El tiempo transcurrido entre estos dos sucesos es el tiempo de servicio (4ms). Observa que el suceso ⑦ planifica el procesamiento de un nuevo cliente si la cola no está vacía.

El diagrama de sucesos para cada uno de los discos es similar al anterior, aunque debemos cambiar el tiempo de servicio que ahora es de 14ms.



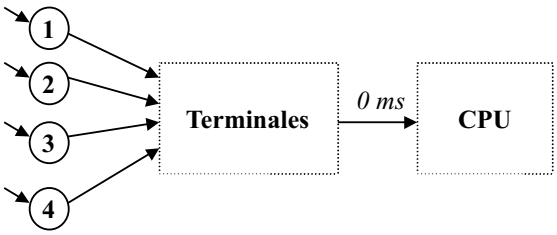
A partir de ahora, en el diagrama de sucesos global omitiremos los detalles de las estaciones de servicio, dibujando solo la caja con el nombre de la estación. Las flechas que apunten a la caja estarán en realidad planificando el primer suceso de la estación (por ejemplo el suceso ⑤ de la CPU). Por otro lado, las flechas que salgan de la caja serán las planificadas por el último suceso de la estación (por ejemplo el suceso ⑦ de la CPU).

Las cuatro terminales del sistema también pueden modelarse como una estación de servicio con cuatro servidores. En este caso no será necesario definir una política de servicio ya que nunca se formarán colas en las terminales. En cuanto al tiempo de servicio, ¿qué representa en las terminales? Dado que una petición de servicio a una terminal representa el inicio de una nueva transacción, el tiempo de servicio de ésta representará el tiempo que transcurre entre dos transacciones distintas (ó el tiempo de preparación de cada nueva transacción). Observa que en la descripción del sistema no se ha contemplado este parámetro; supondremos que éste es de 4000ms. Con todo esto, el diagrama de sucesos de las terminales quedaría como sigue:



Vamos a plantearnos ahora el diagrama de sucesos para todo el sistema. En primer lugar, nos fijaremos en los sucesos iniciales que desencadenan toda la simulación. Éstos serán los que planifiquen las primeras cuatro transacciones que deben lanzarse desde las terminales.

Los cuatro sucesos iniciales planifican el procesamiento en las terminales (suceso ⑭). Como esta estación de servicio tiene cuatro servidores, todas las peticiones serán procesadas simultáneamente, sin que se produzcan colas. A continuación, las terminales lanzan las transacciones que siempre deben comenzar por la CPU. Así pues, la salida de las terminales (suceso ⑮) debe planificar el procesamiento en la CPU (suceso ⑤). Todo lo anterior se representa del siguiente modo:



Observa que en el modelo hemos tomado como unidad mínima de tiempo 1 ms. Todos los tiempos que sean menores de 1 ms se supondrán que valen 0. Así, el tiempo de conexión entre los dispositivos será 0 ms.

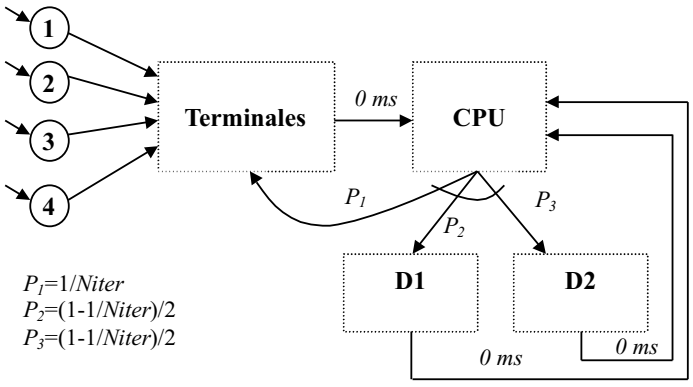
¿Qué sucesos pueden ocurrir a la salida de la CPU? Según la descripción del sistema pueden suceder tres cosas:

1. Que se planifique el final de la transacción, volviendo a las terminales (suceso ⑭)

2. Que se planifique el procesamiento en el disco D1 (suceso ⑧)

3. Que se planifique el procesamiento en el disco D2 (suceso ⑪)

¿Cuándo se seleccionará cada caso? Para responder a esta pregunta es necesario conocer el número medio de interacciones entre los dispositivos durante una transacción. Veamos. Si de media se realizan $Niter$ visitas a la CPU, la probabilidad de terminar una transacción cualquiera será $1/Niter$; y la de seguir la transacción será $1 - 1/Niter$. Dado que el acceso a los discos es equiprobable, esta última probabilidad se dividirá a partes iguales entre los dos discos, es decir que valdrá $(1 - 1/Niter)/2$. Toda esta información se plasma en el diagrama de sucesos como sigue:



Para finalizar el diagrama de sucesos, observa que hemos dibujado dos flechas que partiendo de los discos llegan hasta la CPU. Estas representan el hecho de que siempre hay que volver a la CPU para continuar la transacción.

En la siguiente tabla se resumen todos los sucesos y sus conexiones:

Id	Acción	Vbles. Estado	Planifica
①...④	Inicializa	Todas	14
⑤	Encola petición	CPU.Nq, CPU.W	6
⑥	Comienza servicio	CPU.Ns	7
⑦	Termina servicio	CPU.R	6 y (14, 8 o 11)
⑧	Encola petición	D1.Nq, D1.W	9
⑨	Comienza servicio	D1.Ns	10
⑩	Termina servicio	D1.R	5 y 9
⑪	Encola petición	D2.Nq, D2.W	12
⑫	Comienza servicio	D2.R	13
⑬	Termina servicio	D2.R	12 y 5
⑭	Termina transacción	Ttrans	15
⑮	Comienza transacción	NTrans, Ttrans	5

Ejercicios.

1. Amplia el modelo presentado para tener en cuenta que las *transacciones* pueden acceder además a un servidor remoto con una probabilidad 10 veces menor que a los discos. Supondremos que el tiempo medio para el establecimiento de la conexión remota es de 1000ms, y el tiempo de servicio en el servidor remoto será de 10ms.

2. Modifica el modelo del ejercicio anterior para modelar la situación en la que el tráfico en la red se interrumpe cada 10000ms de media. Cuando esto suceda, todos las peticiones planificadas de conexión al servidor remoto deberán retrasarse en 10ms.

3. Especifica un modelo para una estación de servicio con política *Round Robin*, modificando el modelo básico de una estación de servicio FCFS. Para ello, debes indicar las nuevas variables de estado y dibujar su diagrama de sucesos. Realiza una traza manual del diagrama obtenido, contemplando la posibilidad de que el tiempo de servicio tiende a cero. En este caso estaríamos ante una política de (*Processor Sharing*). ¿Las ejecuciones del modelo propuesto serían eficientes en este caso?

1.4. Bibliografía

M. Fowler "UML distilled a brief guide to the standard modeling language", Addison-Wesley (2000)

D. Ríos Insua et al. "Simulación métodos y aplicaciones", Ed. Ra-Ma, Madrid (1997)

A. M. Law, W. D. Kelton "Simulation Modeling & Analysis", Ed. McGraw-Hill (1984).

R. Jain "The Art of Computer Systems Performance Analysis", Ed. Wiley (1991).

J. J. Pazos et al. "Teoría de Colas y Simulación de Sucesos Discretos", Ed. Pearson/Prentice Hall (2003).