UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

# EXPOSICIÓN TUTORIAL DEL MODELADO Y SIMULACIÓN DE UN ESTABLECIMIENTO DE COMIDA RÁPIDA USANDO ARENA Y SIMKIT

JUAN IGNACIO GIL PALACIOS

Dirigido por: ALFONSO URQUÍA

Curso: 2006-07

EXPOSICIÓN TUTORIAL DEL MODELADO Y SIMULACIÓN DE UN ESTABLECIMIENTO DE COMIDA RÁPIDA USANDO ARENA Y SIMKIT

Proyecto de Fin de Carrera de modalidad oferta general

JUAN IGNACIO GIL PALACIOS

Dirigido por: ALFONSO URQUÍA (firma)

Tribunal calificador:

Presidente: D./Doña. . . . . .
.
(firma)

Secretario: D./Doña. . . . . ..
(firma)

Vocal: D./Doña. . . . . .
(firma)

Fecha de lectura y defensa: . . . . . .

Calificación: . . . . . .

## Exposición Tutorial del Modelado y Simulación de un Establecimiento de Comida Rápida Usando Arena y Simkit


### Resumen

El objetivo de este proyecto es doble: en primer lugar, para evaluar las capacidades y facilidad de empleo de Simkit, estableciendo una comparación entre Simkit (un entorno libre), y Arena (un entorno comercial); y en segundo lugar, para proporcionar material tutorial sobre Simkit y Arena, con el fin de que sea usado en el desarrollo de practicas académicas de simulación.

Se ha realizado el modelado y la simulación de dos modelos utilizando Arena y Simkit. El primero es un modelo simple, cuyo modelado y simulación en ambos entornos se explica detalladamente. El objetivo es proporcionar el material tutorial. El segundo es un modelo complejo de un restaurante de comida rápida, que permite la comparación de las capacidades de Arena y Simkit. Los resultados de simulación obtenidos usando Simkit y Arena son similares.

Una conclusión alcanzada en este estudio es que Simkit puede ser usado satisfactoriamente para el modelado y la simulación de sistemas complejos. La traducción de la representación de la gráfica de eventos del modelo a la descripción que usa Simkit es muy directa. Por estos motivos, Simkit es una opción interesante para el modelado y la simulación tanto de sistemas académicos como complejos.



**Lista de palabras clave:** Arena, Simkit, gráficas de eventos, sistemas de eventos discretos, simulación.


## A tutorial description of the modeling and simulation of a fast food restaurant using Arena and Simkit


### Abstract

The goal of this project is twofold: firstly, to assess the capabilities and easiness-of-use of Simkit, establishing a comparison between Simkit (a free, open source environment) and Arena (a commercial environment); and secondly, to provide tutorial material on Simkit and Arena, intended to be used for self-training on the use of these simulation environments.

The modeling and simulation of two models, using Arena and Simkit, have been discussed. The first one is a simple model, whose modeling and simulation using Arena and Simkit is explained in detail. The goal is to provide tutorial material. The second one is a complex model of a fast food restaurant, which allows comparing the capabilities of Arena and Simkit environments. The simulation results obtained by using Simkit and Arena are similar.

A conclusion achieved in this study is that Simkit can be used successfully for modeling and simulation of complex systems. Even more, the translation from the event graph representation of the model to the description using Simkit is straightforward. For these reasons, Simkit is a right choice for modeling and simulation of academic and complex systems.



**Key words:** Arena, Simkit, discrete event systems, event graphs, simulation.

# ÍNDICE

# Índice de Figuras

# Índice de Tablas

# Capitulo 1.-Introducción, objetivos y estructura del proyecto

## 1.1.-Introducción

El objetivo de la simulación es anticipar el comportamiento de un sistema creando un modelo del mismo. De esta manera, se puede estimar el comportamiento del mismo sin incurrir en los costes de su desarrollo, y decidir previamente la conveniencia o no de realizarlo [Urqu06a].

En el Proyecto realizado se modela el comportamiento de un establecimiento de comida rápida desde dos puntos de vista. Primero usando el paradigma *Orientado a Procesos*, con la herramienta de simulación Arena 7.0 [Kelt02]. Segundo mediante el paradigma *Orientado a Eventos discretos* usando las librerías Java de Simkit [Buss01b]. Se han obtenido resultados similares empleando ambos modelos.

Las fases seguidas para el desarrollo de este Proyecto han sido las siguientes [Urqu06a]:

1. Determinación de las variables y parámetros relevantes para el funcionamiento del sistema real. Es importante elegir de forma adecuada dichos parámetros, ya que si se quiere ser demasiado exhaustivo el modelo será muy complicado y lento en su funcionamiento, y si se es muy simplista el modelo no reflejará el comportamiento relevante del proceso real.

2. Realización de un estudio estadístico de los valores que el entorno del sistema real va a fijar para algunos de dichos parámetros.

3. Análisis de los procesos y módulos adecuados para representar dichas variables y sus relaciones.

4. Realización de fragmentos del modelo que pueden ser probados por separado, para validar su comportamiento.

5. Ensamblado de los fragmentos para constituir el modelo del proceso global.

6. Determinación los parámetros que se pueden variar y de la o las variables a optimizar.

7. Elaboración de un plan de ejecución, obtención de resultados y análisis estadístico de los mismos, con el fin de encontrar la configuración del sistema que conduce a los valores óptimos.

## 1.2.-Objetivos

El Departamento de Informática y Automática de la UNED dispone de una sola licencia para uso universitario de Arena. Los alumnos de la asignatura Simulación (optativa de tercer curso de Ingeniería Técnica en Informática de Gestión) usan una versión educativa, con una capacidad muy reducida, para desarrollar trabajos. Esto impone restricciones en cuanto a la complejidad de los trabajos que pueden realizar los alumnos. En este Proyecto se pretenden evaluar las capacidades y la factibilidad del uso del software libre Simkit, comparando los resultados obtenidos usando Simkit con los obtenidos usando Arena en la simulación de un

sistema complejo, y así mismo se pretende desarrollar material tutorial sobre Arena, y fundamentalmente sobre Simkit, que sirva para que los alumnos puedan aprender el manejo de ambos simuladores.

## 1.3.-Estructura

Esta memoria se ha estructurado en ocho capítulos. En el Capítulo 1 se describe de forma general el contenido del trabajo realizado, así como los objetivos y estructura del Proyecto.

En el Capítulo 2 se describen las fases generales de un proceso de modelado, dos paradigmas de modelado junto con las dos herramientas: Arena y Simkit basadas en ellos. Se detalla el manejo de dichas herramientas, ilustrándolo mediante el desarrollo con ambas de un modelo sencillo (modelo simple).

En el Capítulo 3 se describe de forma detallada el modelo de un sistema complejo (un establecimiento de comida rápida).

En el Capítulo 4 se muestra como se ha realizado el modelo descrito en el Capítulo 3 utilizando Arena. Se realiza una descripción detallada de los conjuntos de bloques de construcción que se han utilizado. Las características del modelo que simula un establecimiento de comida rápida hacen que se precise la versión completa de Arena para su modelado.

En el Capítulo 5 se describe la utilización de Arena para realizar simulaciones con el modelo construido. Se recogen los parámetros de entrada usados y los resultados obtenidos. Se establece un criterio de optimización. Se encuentran los parámetros que producen el resultado óptimo y el valor de éste.

En el Capítulo 6 se discute detalladamente la construcción del modelo descrito en el Capítulo 3 utilizando las librerías de Simkit. Se parte de la gráfica de eventos, que conduce de forma directa a la implementación en lenguaje Java de la clase central de la simulación (extensión `simkit.SimEntityBase`). Para hacer más fácil la interacción del modelo con el usuario, se ha implementado de una GUI (interfaz gráfica de usuario) que gestiona la interacción del usuario con dicha clase, permitiendo configurar diferentes parámetros y modos de ejecución así como la posibilidad de guardar los resultados.

En el Capítulo 7 se describen los resultados de la ejecución del modelo Simkit con distintos parámetros de entrada. Se ha preparado un juego de parámetros de ensayo (que incluyen los probados por el modelo Arena). Dicho juego es leído por la aplicación Simkit al lanzarla en modo optimización y se pueden guardar los resultados obtenidos para análisis posteriores. La aplicación selecciona los parámetros que conducen a resultados óptimos. El resultado de la ejecución del juego de parámetros de ensayo (1000 ejecuciones de dichas combinaciones) se ha ilustrado recogiéndolo en diversas gráficas incluyendo aquellas en que se comparan los resultados obtenidos por Arena y Simkit para los mismos juegos de parámetros. Los resultados para configuraciones iniciales iguales de los modelos de Arena y Simkit producen resultados equivalentes.

En el Capítulo 8, se proponen líneas de trabajo futuras para hacer más fácil y productivo el uso de las librerías Simkit.

## 1.4.-Estructura del C.D-ROM.

En el C. D. se han incluido los cuatro directorios mostrados en la Figura 1.1.


**Fig. 1.1**: Carpetas del CD-ROM.

1. *ProgramasJava* incluye las dos aplicaciones Java que se han desarrollado en este Proyecto junto con las librerías Simkit que ambas han usado (ver figura 1.2).

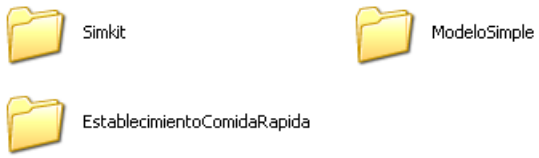


**Fig. 1.2**: Carpetas de ProgramasJava

1.1 *modeloSimple* contiene la aplicación sencilla que ha servido para ilustrar de forma detallada el modelado con Simkit, en dicha carpeta podemos encontrar los

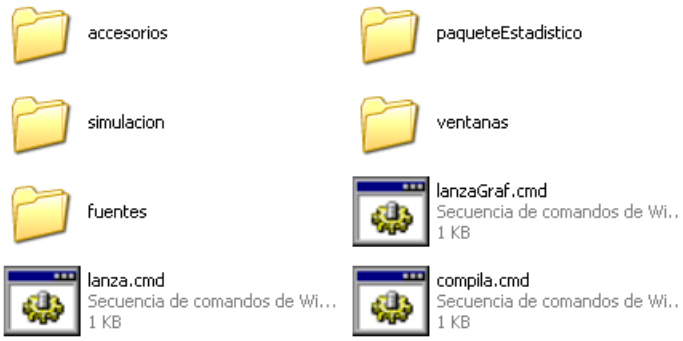


**Fig. 1.3**: Carpetas de ModeloSimple.

archivos y carpetas de la figura 1.3

a. *accesorios* con las clases ejecutables de dicho paquete que se ocupan de tareas como análisis de patrones textos, leer archivos y guardar archivos.

b. *paqueteEstadistico* con una clase ejecutable que genera una distribución estadística (exponencial con parámetro variable en el tiempo de forma periódica).

c. *simulacion* con las clases ejecutables de dicho paquete que realizan las tareas de simulación.

d. *ventanas* con las clases ejecutables de dicho paquete que generan la GUI.

e. *fuentes* contiene los archivos fuente Java de los paquetes de modelo simple cuyos ejecutables están en las cuatro carpetas anteriores.

f. *lanza.cmd "script"* que permite ejecutar el modelo simple sin interfaz gráfica de usuario.

g. *lanzaGraf.cmd "script*" que permite ejecutar el modelo simple con la interfaz gráfica de usuario que se ha diseñado con ella.

h. *compila.cmd "script*" que permite compilar los programas Java de la carpeta fuentes para obtener los ejecutables incluyéndolos en una carpeta que se llamará *ejecutables*.

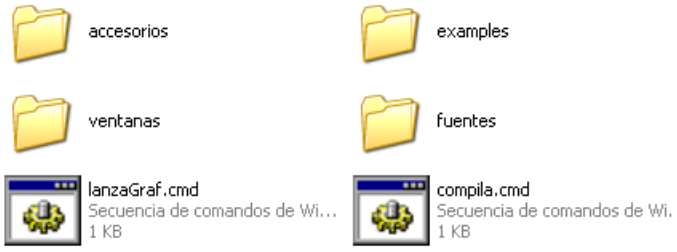


**Fig. 1.4**: Carpetas de EstablecimientoComidaRapida

1.2 *EstablecimientoComidaRapida* contiene la aplicación que modela un establecimiento de comida rápida usando las librerías de Simkit y Java (ver Figura 1.4)

a. *accesorios* con las clases ejecutables de dicho paquete que se ocupan de tareas como análisis de patrones textos, leer archivos y guardar archivos.

b. *examples* con las clases ejecutables de dicho paquete que se encargan de gestionar los procesos de la simulación del establecimiento de comida rápida.

c. *ventanas* con las clases ejecutables de dicho paquete, que se encargan de mostrar la GUI.

d. *fuentes* con los ficheros fuentes de los tres paquetes comentados anteriormente.

e. *lanzaGraf.cmd "script*" que permite ejecutar el modelo del establecimiento de comida rápida.

f. *compila.cmd* "*script"* que permite compilar los programas Java de la carpeta fuentes para obtener los ejecutables incluidos en una carpeta que se llamará *ejecutables*.

1.3 *Simkit* contiene las librerías usadas para el desarrollo de los modelos Java. Están contenidas en dos archivos jar: *actions.jar* y *simkit.jar*

2. *ModeloArena* incluye los archivos del modelo del establecimiento de comida rápida desarrollado con Arena. El listado fuente de las funciones Visual Basic auxiliares se ha incluido en un archivo de la carpeta *fuentes Visual Basic* [Ceba06] (ver Figura 1.5).

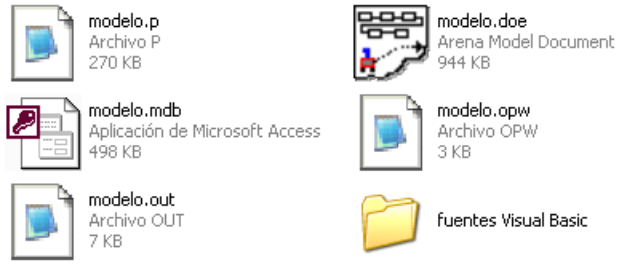


**Fig. 1.5**: Contenido de la carpeta ModeloArena

3. *Resultado Optimizacion* contiene dos carpetas con los resultados de las optimizaciones realizadas con Arena y Simkit y que luego se han comentado en la memoria (ver Figura 1.6).


**Fig. 1.6**: Carpetas de ResultadoOptimización.

3.1 *Arena* contiene archivos excel con los datos ensayados y resultados obtenidos en el proceso de optimización realizado con el modelo de Arena.

3.2 *Simkit* contiene archivos de texto con datos ensayados y resultados obtenidos en el proceso de optimización realizado con el modelo de Simkit.

4. *Memoria* contiene dos versiones de la memoria del Proyecto: *memoria.wpd* es la versión desarrollada con WordPerfet12 y *memoria.pdf*, con el mismo, contenido pero en formato pdf.

## 1.5.-Conclusiones

La experiencia acumulada en la realización de los trabajos descritos en esta memoria nos permite afirmar que las librerías Simkit pueden prestar apoyo didáctico para tareas de modelado.

Como se describe en esta memoria Simkit tiene potencia como para poder ser empleado en el desarrollo de modelos de elevada complejidad.

El tiempo de desarrollo de un modelo complejo con Simkit no ha sido mucho mayor que la realización con Arena.

Aunque Arena tiene módulos estándar que se ensamblan fácilmente, cuando el modelo precisa un número grande de ellos, resulta complicado comprobar que un conjunto grande simula de forma correcta el elemento del modelo al que representan (sobre todo si no hay bloques que se adapten y se requiere del uso de varios). Con Simkit se puede crear el objeto con el comportamiento deseado de forma directa utilizando Java, se puede probar de forma separada y se puede reutilizar sin problemas.

No es difícil aprender el uso de los paquetes básicos de dichas librerías.

Aportan la flexibilidad de estar basadas directamente en Java, un lenguaje de propósito general.

El hecho de que la implementación del diagrama de eventos futuros con estas librerías sea bastante inmediata, puede servir de apoyo didáctico para la comprensión de dichos diagramas, sin precisar un gran esfuerzo de programación para desarrollar programas que los implementen.

No parece difícil ampliar dichas librerías con paquetes propios, para dotarlas de elementos de uso frecuente en los modelos que no se han incluido en ellas de forma estandarizada, como son entidades, recursos, colas etc.

# Capítulo 2.-Modelado y simulación de sistemas logísticos y de producción

## 2.1.-Introducción

En este Capítulo se va ha realizar una descripción detallada de las fases a seguir para realizar un proceso de modelado [Guas02]. Se describen con detalle los elementos de dos de los paradigmas (*orientado al proceso* y *orientado a eventos discretos*) y dos herramientas basadas en dichos paradigmas Arena y Simkit respectivamente. Para ilustrar la descripción se implementa con las dos herramientas un modelo sencillo, pero que precise utilizar al menos los elementos básicos de ambas. Se describen con gráficos y en detalle los elementos utilizados y su configuración.

## 2.2.-Fases de un estudio mediante simulación

Las fases a seguir se describen brevemente a continuación [Brat87] :

1. **Definición del problema y planificación del Proyecto**. Consiste en definir claramente las razones para estudiar el sistema y los objetivos que se persiguen. También se deberá hacer un estudio preliminar de las necesidades (hardware, software y mano de obra) para el desarrollo del modelo. Las razones para el desarrollo de un modelo pueden ser de diversa índole, pudiéndose destacar: evaluación del diseño de sistemas, comparación de

dos o más diseños o políticas, predicción del comportamiento de un sistema en determinadas circunstancias, búsqueda de optimización de procesos, búsqueda de relaciones funcionales entre variables que caracterizan al proceso real, localización de puntos críticos y cuellos de botella etc. El propósito del estudio determina en gran manera el diseño del modelo, pues no todas las razones para el desarrollo de modelos requieren de representaciones con el mismo nivel de precisión.

2. **Definición del sistema y formulación del modelo**. Consiste en identificar un pequeño conjunto de características o propiedades del sistema que sean suficientes para determinar con suficiente fidelidad su comportamiento. A grandes rasgos se puede seguir la siguiente metodología: escoger las variables de salida; determinar los componentes del sistema que influyen en dichas variables y decidir si dicha influencia va a ser parte del modelo o se va a considerar externa, en este caso será un parámetro de entrada; determinar la lógica del modelo estableciendo las relaciones entre las variables de entrada hasta llegar a las de salida de forma que reflejen el comportamiento del sistema real.

3. **Diseño de los experimentos.** Se realiza en dos fases. Primeramente en la fase de desarrollo con el objetivo de facilitar el mismo ayudando a verificar y validar el modelo. Una vez completado el modelo, se revisa el diseño experimental y se orienta a obtener las respuestas para las que se construyó el modelo.

4. **Datos de entrada**. Es importante la elección de los datos de entrada. Normalmente éstos pueden tener dos tipos de procedencia:

    a. Datos experimentales del sistema, pudiéndose tomar directamente del mismo o usar distribuciones estadísticas de datos previamente muestreados;

b. Datos teóricos obtenidos de personal familiarizado con el funcionamiento de sistemas semejantes. Esta opción se toma cuando el sistema a simular no existe o por que no es posible obtener los datos.

5. **Traducción del Modelo**: Es el proceso por el que el modelo teórico se convierte en un modelo informático. Este proceso se ha visto facilitado por la evolución de los lenguajes de programación, la aparición de librerías orientadas a la simulación y por la aparición de entornos de modelado que facilitan el desarrollo de modelos (GPSS, SIMSCRIPT, SLAM, SIMAN ). En este Proyecto se ha utilizado dos entornos diferentes Arena basado en SIMAN [Kelt02] , y Simkit [Buss01a] que consiste en un conjunto de librerías Java por lo que se puede integrar en entornos de desarrollo de aplicaciones Java como Eclipse.

6. **Verificación y validación del modelo**: la verificación consiste en comprobar que no hay errores en la traducción del modelo a instrucciones del programa. La validación consiste en comprobar que el modelo representa de forma suficientemente fiel la realidad. Para realizar la verificación se pueden usar algunos de los siguientes mecanismos: verificación manual de la lógica, comprobación submodelo a submodelo, comprobación de soluciones conocidas y realización de tests de sensibilidad.

7. **Experimentación y análisis de los resultados**. Normalmente requieren de planificaciones estadísticas. Los experimentos de simulación suelen tener uno de estos dos comportamientos: existe una condición clara de terminación para el proceso de simulación o por el contrario no existe dicha condición y la simulación es sin terminación prolongándose en este caso el tiempo necesario hasta alcanzar resultados independientes de los parámetros iniciales es decir hasta alcanzar un estado estacionario.

8. **Documentación e implantación**: son la culminación de un proceso de simulación. Éste no puede considerarse completo hasta que no se ha documentado el proceso de desarrollo y los resultados, no se han estudiado e interpretado los mismos y no se ha obrado en consecuencia. Una documentación adecuada alargará la vida útil del modelo e incrementara la posibilidad de reutilización de todo o partes del mismo.

## 2.3.-Paradigmas de modelado

La experiencia acumulada en el modelado de procesos ha permitido determinar la existencia de un conjunto de elementos que aparecen con frecuencia. Se pueden aprovechar las estructuras creadas para representar esos elementos para ser reutilizadas en el modelado de otros procesos diferentes. Un camino alternativo es plantearse el proceso a modelar como una secuencia de eventos que ocurren con cierto orden temporal y que provocan o evitan la ocurrencia de otros eventos posteriores. Estos enfoques no son completamente excluyentes, sino que muchos entornos basados en bloques reutilizables internamente funcionan como una secuencia de eventos, y modelando mediante la planificación de eventos se encuentra que es de utilidad disponer de modelos reutilizables de objetos que frecuentemente aparecen en el modelado de procesos. A continuación, se describen con mayor detalle ambos paradigmas.

1. **Orientado al proceso**. En los modelos de simulación se observa que las abstracciones del sistema siguen tipos de procesos: colas, retrasos, bifurcaciones, fusiones, toman o liberan recursos etc. Este paradigma proporciona módulos estándar que simulan el comportamiento de dichos tipos de procesos. El trabajo del diseñador consiste en descomponer el modelo global en un conjunto de esos módulos estándar y proceder al ensamblado de los mismos [Kelt02].

2. **Orientado a los eventos discretos.** [Cass99] Encontrar los entes que representen al sistema y establecer una lista ordenada temporalmente de los eventos que van a ocurrir con dichos entes. La ocurrencia de un evento ocasiona un tratamiento, la posible planificación de uno o más eventos futuros que se deberán incluir en la lista en las posiciones que les correspondan y/o bien la posible revocación de eventos planificados y no ocurridos. La simulación finaliza cuando se vacíe la lista de eventos futuros o bien fijando alguna condición [Buss01a].

## 2.4.-Modelado mediante gráficas de eventos

La simulación se inicia con un evento inicial que planifica los primeros eventos del modelo propiamente dicho y continúa hasta que la lista de eventos futuros queda vacía o se alcanza alguna condición prefijada de fin de simulación. El tiempo de simulación avanza con

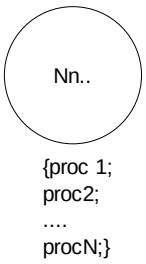


**Fig. 2.1**: Ejemplo de nodo.

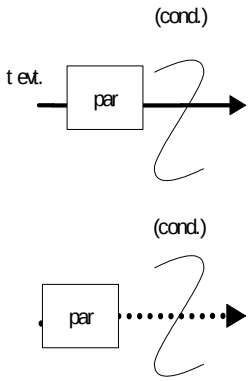


**Fig. 2.2**: Ejemplos de planificar y revocar

incrementos discretos que vienen dados por el tiempo planificado para el próximo evento en tratamiento. Como el nivel de abstracción de este paradigma es menos elevado que el *orientado a procesos* es conveniente usar metodologías gráficas para el modelado de sistemas, una particularmente adaptada a este paradigma es "Gráficas de Eventos" (*"Even Graph"*) que de forma minimalista permite realizar representaciones de este paradigma [Buss01a].

Los elementos gráficos son nodos que se representan como un círculo (véase la Figura 2.1 ) que tiene como atributos, el nombre del nodo (Nn..) y un conjunto de procesos (proc1,...procN) que se representan entre llaves debajo del círculo. De ellos pueden salir una o mas flechas de 2 tipos que los conectan con otros nodos (véase la Figura 2.2). El nombre identifica el tipo de evento que representa el nodo, proc1... procN representan los procesos que se realizan cuando ocurre dicho tipo de evento y las flechas continuas conectan con los eventos futuros que se planifican, al tiempo que ocurre el tratado, mientras que las líneas discontinuas conectan con los eventos planificados que se revocan en dicho evento.

Las flechas (*"edges"*) de línea continua tienen como parámetros:

1. "$t_{evt}$" expresión numérica que indica el tiempo para el que se planifica el evento del tipo que indica el destino de la flecha.

2. "(cond.)" Expresión lógica que indica si se planifica o no el evento de destino de la flecha.

3. "par" parámetro que se pasa al evento destino de la flecha.

Las flechas discontinuas tienen un parámetro menos y su significado es el siguiente:

1. "(cond.)" expresión lógica que indica si se revocará o no un evento del tipo del destino de la flecha.

2. "par" parámetro que se pasa para la revocación del evento destino de la flecha discontínua.

### 2.4.1.-Ejemplo

En la Figura 2.3. se muestra un diagrama que representa el modelo descrito a continuación. Hay tres tipos de eventos representados por “A”, “B” y “C”. Si se trata “A” se incrementa la variable “h” en 5 unidades y se planifica “B” con el parámetro “j” para dentro de 3 unidades de tiempo si “n>5”; se planifica incondicionalmente y sin parámetro “A” para un tiempo igual a “h”; y se revoca un evento de tipo “C” pasando como dato “k” siempre que h>3.

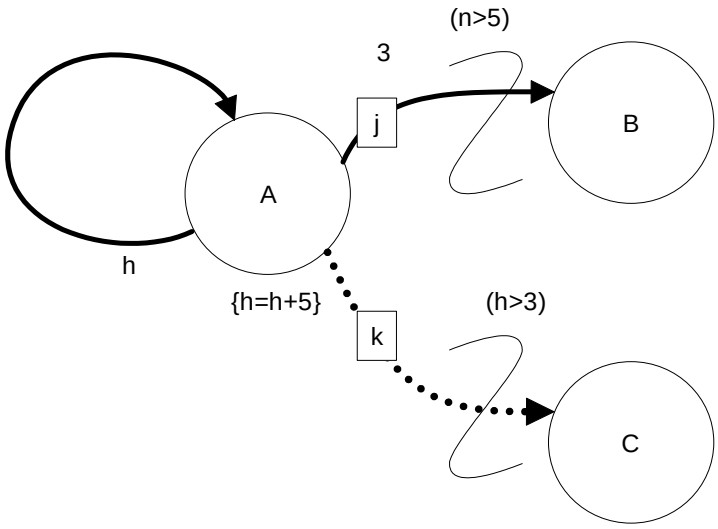


**Fig. 2.3**: Diagrama de eventos sencillo.

## 2.5.-El entorno de simulación Arena

Arena es un entorno gráfico que asiste en la implementación de modelos en el paradigma orientado al proceso. Contiene un conjunto de bloques que representan gráficamente los diversos tipos de procesos agrupados en categorías en función de su complejidad. Los bloques se enlazan para representar el flujo de entidades entre ellos. Asiste para representar gráficamente la animación del proceso, pudiendo fijar el aspecto gráfico de las entidades que fluyen. También

permite fácilmente crear cuadros en los que se muestren los valores de variables del modelo según transcurre la simulación. Recopila datos de estadísticas de la simulación; los cuales se pueden visualizar mediante informes al final de la misma [Kelt02]. Permite incluir módulos de usuario elaborados en otros lenguajes de programación, tales como FORTRAN, C o Visual Basic. Con este último tiene una fácil integración, ya que contiene una herramienta que permite editar, compilar y enlazar código realizado en este lenguaje. Tiene incorporadas librerías Visual Basic que permite presentar o tomar datos de hojas de cálculo Microsoft Excel.

Para el diseño tiene 14 paneles de modelado:

"Basic Process", "Advanced Process", "Blocks", "Advanced Transfer", "Contact Data", "CSUtil", "Elements", "Factory Elements", "Packaging", "Factory Blocks", "AgentUtil", "Factory", "Script" y "UtlArena".

Los más generales y que se pueden usar sin restricciones en la versión completa son los 4 primeros. El panel que ofrece por defecto es "Basic Process", que contiene bloques para un amplio rango de elementos frecuentes en modelados de forma que en esté Proyecto casi todos los bloques usados pertenecen a este panel. "Blocks" son bloques de construcción más elementales que los del panel "Basic Process". La funcionalidad que ofrecen los elementos del "Basic Process" requiere, si se quiere modelar con elementos de "Blocks", del ensamblaje de varios elementos. Los elementos de "Blocks" son de utlilidad cuando no hay elementos simples de "Advanced Process" que modelen el comportamiento deseado y sin embargo ensamblando elementos más simples de "Blocks", se pude conseguir. En el Proyecto descrito se han utilizado unos pocos elementos de "Blocks" para modelar el establecimiento de comida rápida (ver Figura 4.3 y Figura 4.15). Como ejemplo sencillo de su utilidad se puede citar lo el siguiente ejemplo, en "Basic Process" no hay un bloque sólo "seize", si se precisa este comportamiento se puede

tomar el bloque “seize” de “Blocks”. “Advanced Process” y “Advanced Transfer” tienen elementos que modelan comportamientos más complejos que “Basic Process”. El resto de los paneles contienen elementos de modelado orientados a tareas específicas y tienen ciertas restricciones de uso incluso en la versión completa.

La duración de la simulación se puede fijar de forma temporal o bien indicando una condición. Se puede lanzar la simulación con animación o en modo *Batch,* en que la animación se suspende y la animación es mucho más rápida.

Tiene una aplicación que puede realizar optimizaciones automáticas en el modelo “Optquest for Arena”. Se puede acceder a ella en la opción de menú “Tools”. Una vez iniciada permite configurar la optimización en modo gráfico y luego permite lanzar el proceso.

Cuando se lanzan las simulaciones, Arena recopila de forma automática estadísticas sobre los distintos elementos del modelo, entre las que podemos destacar las de recursos y colas. De los recursos obtiene de forma automática, el porcentaje de utilización, el número total de veces que se han usado durante la simulación y el número de unidades puestas en juego. De las colas obtiene los tiempos mínimos, medios y máximos de espera; el número mínimo, máximo y promedio de entidades esperando. También admite que el usuario pueda definir estudios estadísticos.

## 2.6.Descripción de un modelo sencillo usando Arena

A continuación, se desarrolla un modelo simple usando Arena [Urqu06b]. El modelo simula una cadena de empaquetamiento de componentes. Los componentes se generan como entidades y tras su llegada (creación) al centro de empaquetado, se chequea si es erróneo. Si el componente es erróneo se envía a una cadena auxiliar de reparación, tras la cual se devuelve a la cadena

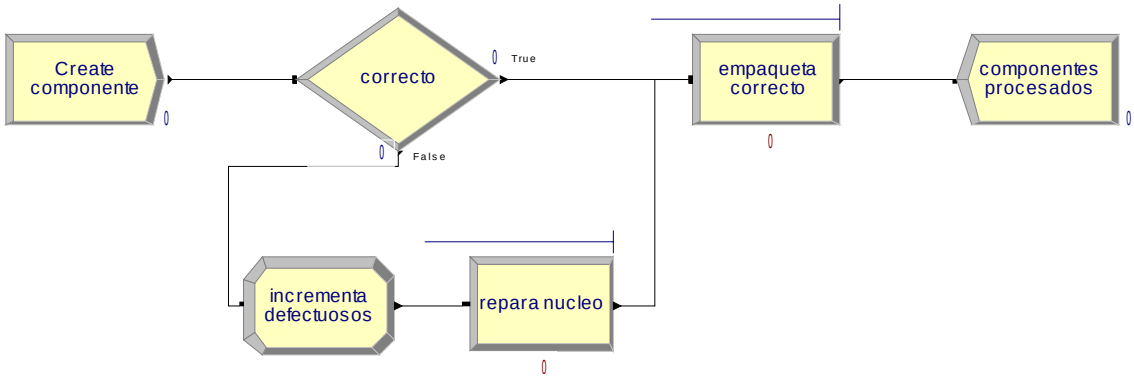


**Fig. 2.4**: Diagrama de bloques del modelo simple.

principal para su empaquetado. El modelo completo se simula mediante los bloques de la Figura 2.4.

El primer bloque “Create componente”, es un bloque de tipo “Create”, que tiene como finalidad generar entidades. En nuestro ejemplo se utiliza para simular la llegada de objetos a la cadena de empaquetado. Es necesario configurarlo con una función estadística que simule el ritmo de llegada real. Para nuestro ejemplo hemos supuesto una distribución exponencial de tiempos con parámetro 1, la unidad de medida se ha tomado el minuto y el número de

componentes que llegan juntos es 1, no se limita el número de total de entidades que se pueden generar en el transcurso de la simulacióny la primera entidad se genera para tiempo 0.0. La

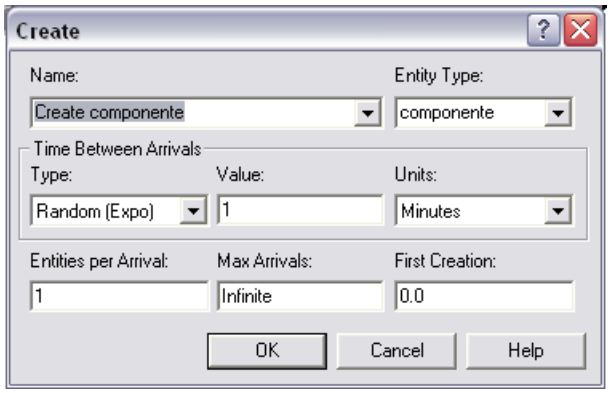


**Fig. 2.5**: Diálogo de Create Componente.

configuración de este bloque se muestra en la Figura 2.5

Cambiando los parámetros se pueden simular otras distribuciones, etc.

El segundo bloque “correcto”, es un bloque de tipo “Decide”, que dirige las entidades que entran en él a una de sus dos salidas (o más salidas, en el ejemplo sólo se dispone de dos salidas). La elección se puede hacer al azar, o bien dependiendo de algún atributo de la entidad, (en el ejemplo se hace al azar, 10% va a la salida “false” y 90% a la “true”, es decir un 10% de los componentes son defectuosos y se derivan a la rama de reparación). La configuración de los bloques “Decide” se hace en una ventana del tipo de la mostrada en la Figura 2.6.

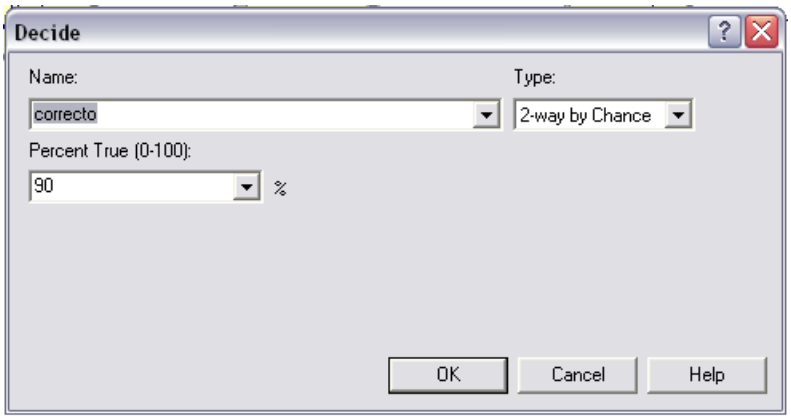


**Fig. 2.6**: Dialogo de Decide.

En dicha ventana se puede editar el nombre del bloque, se puede elegir el tipo (2-way by Chance), de dos salidas se toma una al azar, aunque se pueden elegir otros tipos como (2-way by Condition, N-way by Chance y N-way by Condition) que corresponden que de dos salidas se toma una según se cumpla o no una condición, de N salidas se toma una al azar, y de N salidas se toma una según se cumpla o no una condición. Si se eligen al alzar, queda por indicar los porcentajes asignados a cada salida en nuestro caso 90% a true. En los casos en que se elige una condición se indica la expresión de dicha condición.

El bloque que sigue a la salida true es un bloque "Process", "empaqueta correcto", en este tipo de bloque se simula un procesamiento de la entidad que llega a él. En nuestro ejemplo se simula el empaquetado del componente. El proceso a realizar puede consistir en un retraso temporal; un retraso temporal y la ocupación de un recurso; un retraso temporal, la ocupación de un recurso y su liberación; o bien un retraso temporal y la liberación de un recurso ocupado

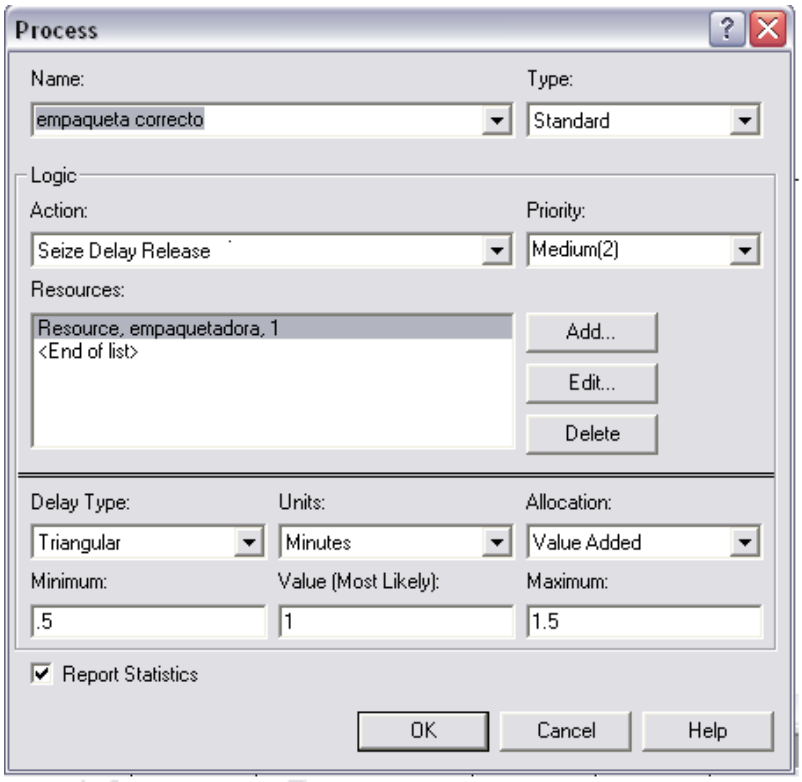


**Fig. 2.7**: Diálogo de Process.

previamente, que corresponden a las opciones "Delay", "Seize Delay", "Seize Delay Release" y "Seize Delay Release" respectivamente de una de sus opciones de configuración. La ventana de configuración de dicho bloque se muestra en la Figura ?.

Se comienza indicando el nombre asignado al bloque.

En Type se puede indicar si se trata de un simple bloque (Standard) o bien el proceso a simular es más complejo y precisa descomponerlo en más bloques que constituirían un submodelo (Submodel).

En action se indica el tipo de procesamiento a realizar, que puede ser uno de los cuatro tipos ya indicados ("Delay", "Seize Delay", "Seize Delay Release" y "Seize Delay Release" ).

En Priority se indica para el caso de que se ocupen recursos la prioridad con que cuenta este bloque frente a otros bloques en que se ocupen los mismos recursos, se pueden indicar tres niveles (High(1), Medium(2) y Low(3)).

En Resources se añaden líneas. En cada línea se incluye una expresión que indica el número de recursos de un tipo que se van a ocupar y/o liberar. Con los botones Add, Edit y Delete se pueden añadir líneas, editar la seleccionada o borrar la seleccionada respectivamente. Ya sólo queda configurar el tipo de retraso que ocasiona el procesamiento, se debe elegir la distribución para dicho retraso, la unidad temporal en que se mide; una vez hecho esto se procede a configurar los parámetros necesarios para dicha distribución temporal.

La ventana Allocation indica como se debe imputar a la entidad el coste y el retraso del procesamiento. En la ventana de la Figura ?, se muestra la configuración del ejemplo utilizado Name: empaquetado correcto; Type: Standard; Action: Seize Delay Release; Priority:

Medium(2); Resources: Resource, empaquetadora, 1; Delay type: Triangular; Units: Minutes; Allocation: Value Added; Minimun: 0.5; Value(Most Likely): 1; Maximun: 1.5. Esta configuración indica lo siguiente: Se ocupa una unidad del recurso empaquetadora, produciéndose un procesado de ocupación espera y liberación del recurso, la prioridad de ocupación es media (en este modelo no tiene importancia ya que el recurso solo se usa en este bloque), la distribución temporal de la espera es una distribución triangular con mínimo 0.5 minutos, máximo 1.5 minutos y moda 1 minuto.

El bloque que sigue es un bloque “Dispose” “Componentes procesados”. Su cometido es eliminar las entidades que llegan a él ya que ya no van a ser necesarias, hace un recuento de las que van llegando. Su configuración es sencilla pues sólo precisa que se le asigne un nombre, (ver la Figura 2.8).

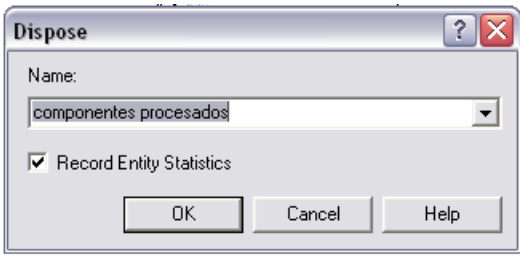


**Fig. 2.8**: Diálogo de Dispose.

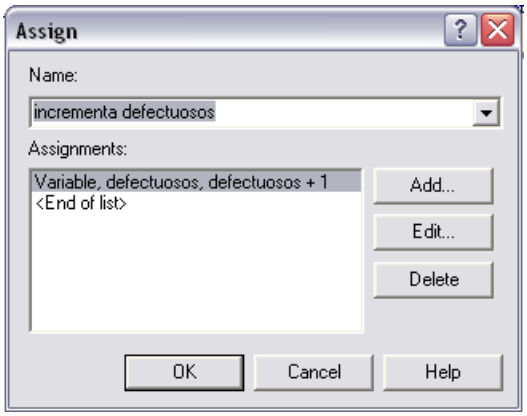


**Fig. 2.9**: Diálogo de Assign.

El bloque “Assign” “Incrementa Defectuosos” situado en la salida false del bloque “Decide” “correcto” tiene como cometido incrementar la variable global que contabiliza los componentes defectuosos: “defectuosos”. La configuración de este tipo de bloque es sencilla pues consiste en asignarle un nombre e incluir las líneas necesarias en las que se incluirán expresiones que alterarán los valores de las variables globales o de los atributos de las entidades que pasan por él. Véase la Figura 2.9.

En la Figura 2.9 se muestra la configuración del bloque “Assign” del ejemplo, en él se altera el valor de la variable “defectuosos” incrementándose su valor en una unidad al paso de cada entidad defectuosa.

El único bloque que queda es un bloque de tipo “Process” “repara núcleo” como ya se han explicado sus características con anterioridad tan solo comentaremos los valores que se le han dado en el ejemplo ver la Figura 2.10.

El bloque se ha configurado con captura de una unidad del recurso “corrector”, su uso durante un tiempo distribuido triangularmente con mínimo de 1.5 minutos, máximo de 5.5

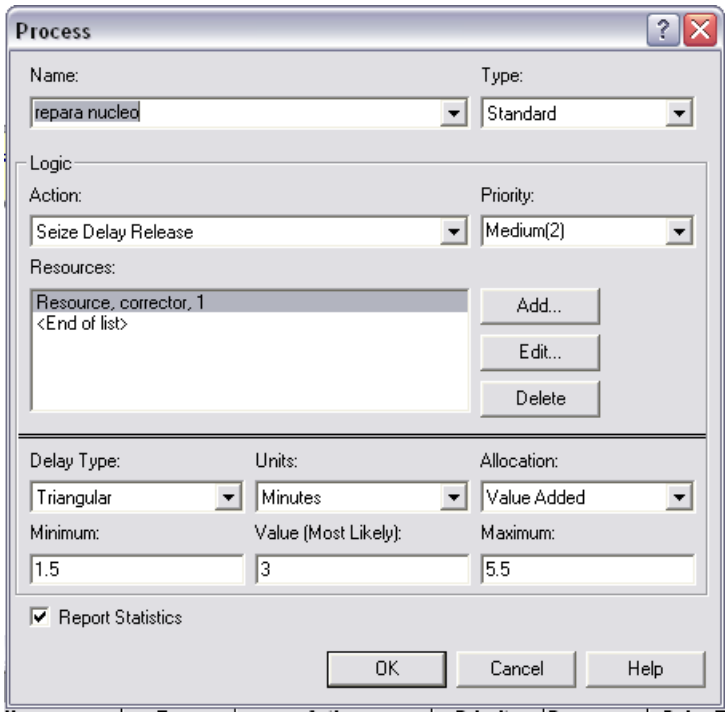


**Fig. 2.10**: Diálogo de Process.

minutos y moda de 3 minutos; y su posterior liberación, el tiempo y coste del procesado se añade a la entidad y la prioridad de ocupación del recurso es media.

### 2.7.-Las librerías de simulaciónde Simkit

Simkit es un conjunto de librerías Java [Buss01b] que facilitan la gestión de la "lista futura de Eventos", pieza central del paradigma orientado a eventos y facilita también la gestión del tiempo de simulación y del número de eventos ocurridos. Tiene librerías para obtener distintas distribuciones de probabilidad, para generar estadísticas, un paquete de ejemplos, dos paquetes para simular combates, un paquete de utilidades y un paquete experimental para interaccionar con ficheros XML. A continuación, se detallan los nombres de los paquetes, una información mucho más amplia se puede obtener leyendo los documentos html que se incluyen en la carpeta doc que se genera en la carpeta de instalación de Simkit.

1. `simkit`: paquete básico que aporta la funcionalidad necesaria para traducir diagramas de eventos a Java.

2. `simkit.ramdom`: paquete que contiene las clases que generan las distintas funciones aleatorias y elementos asociados.

3. simkit.stat: paquete que contiene clases que permiten generar estadísticas de parámetros o resultados de la simulación.

4. `simkit.examples`: contiene ejemplos de implementación de modelos sencillos en Simkit.

5. `simkit.smd` y `simkit.smdx` dos paquetes, el primero está previsto que desaparezca. Tienen clases para simular modelos de combate.

6. `simkit.util`: conjunto de clases con diversas utilidades.

7. `simkit.xml`: paquete experimental, para usar xml en combinación con estas librerías.

El paquete, lógicamente, más importante es el primero ya que en él se define el modelo estándar de función para realizar el tratamiento de un evento y planificación de los que en el se originen, y gestión de la lista de eventos. Con dicho esqueleto se puede, añadiendo el código Java, construir cualquier tipo de modelo. En los modelos desarrollados en este trabajo se han utilizado clases de los tres primeros paquetes de la lista. El 4º paquete (`simkit.examples`) es muy útil para comenzar a usar Simkit en casos sencillos, ya que contiene ejemplos sencillos ya desarrollados.

## 2.8.-Descripción de un modelo sencillo usando Simkit

A continuación se desarrolla el ejemplo comentado para Arena utilizando los diagramas gráficos de eventos de la Figura 2.11 y las librerías de Simkit.

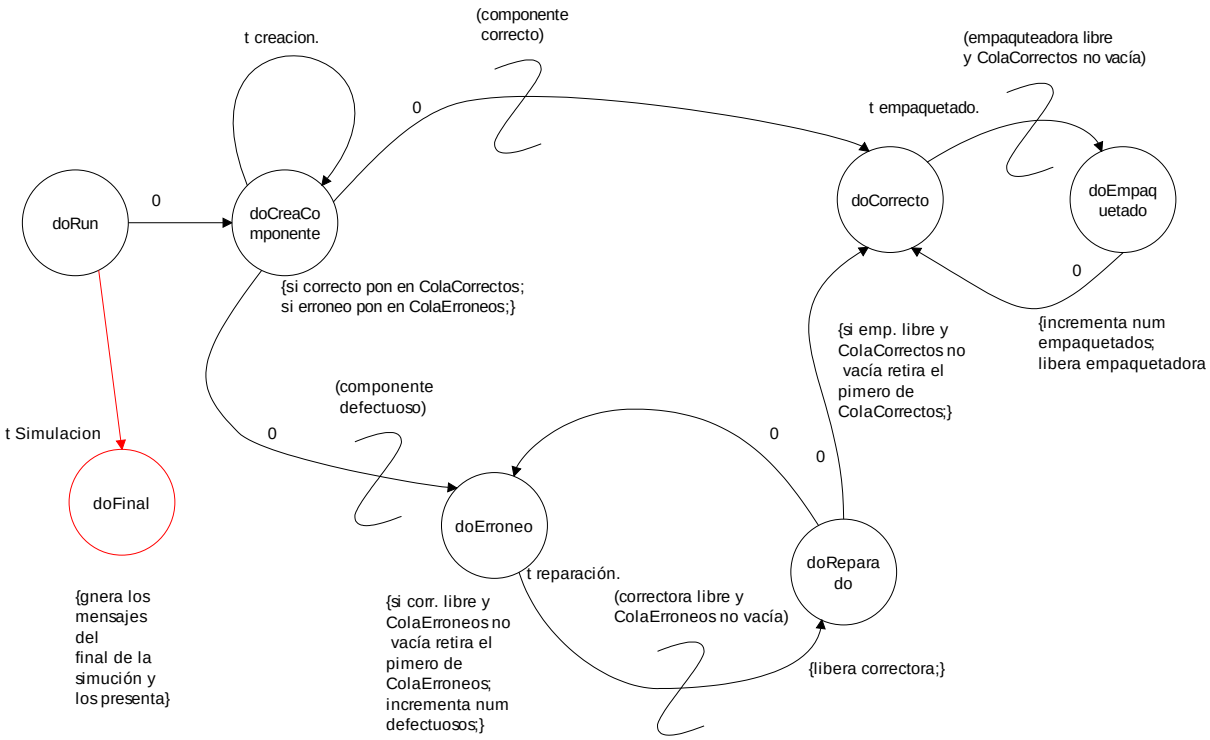


**Fig. 2.11**: Gráfico de Eventos del modelo simple.

Simkit [Buss01b] proporciona un mecanismo simple para traducir el diagrama a código Java, extendiendo la clase `simkit.SimEntityBase` del primer paquete, a cuyas extensiones se pueden añadir métodos que gestionen los distintos eventos discretos que van a darse en el modelo. Para crear uno de dichos métodos basta con anteponer "do" al nombre previsto para el evento (en el diagrama se les ha antepuesto do). Simkit utiliza la "reflection" para encontrar el método adecuado para el nombre usado en la lista de eventos planificados ( utilizando las clases de `java.lang.reflect` se pueden obtener objetos de la clase `Method` con referencias a los métodos de cualquier clase. A partir de los objetos `Method` se pueden conocer los nombres de los métodos

e incluso invocarlos). Si no lo encuentra simplemente no ejecuta el código y pasa al siguiente evento planificado. La lista de eventos es gestionada por la clase `simkit.Schedule` (también del paquete `simkit`), que controla las variables de la simulación como el tiempo, número de eventos etc. Esta clase contiene métodos que nos permiten iniciar la simulación o fijar las condiciones para su finalización [Buss01b].

La tarea del implementador en este momento sería crear la clase que extiende a `simkit.SimEntityBase` e implementar en Java los métodos que corresponden a los eventos del diagrama. Esto se ha hecho creando la clase `simulacion/Nucleo.java`.

No existe implementación en Simkit que soporte objetos como las entidades, recursos etc., sino que el propio diseñador debe decidir qué tipo de objeto Java es conveniente de acuerdo con la simulación a realizar.

Como se ha partido de modelos realizados en Arena, se ha creado una clase `simulacion/Entidad.java` que permite contener algunas de las propiedades de las entidades que usa Arena. Esta clase sería la clase a extender por las entidades concretas a usar por el modelo. En nuestro caso, Entidad se extiende para originar `simulacion/Componente.java`. De esta forma, se puede extender en distintas clases para representar más de un tipo de entidad en un modelo. La clase general Entidad maneja atributos como: un número de serie, el tiempo de creación y una tabla de distribuciones estadísticas (donde se incluyen referenciadas por el nombre de la clase las distribuciones estadísticas de las extensiones concretas de la clase Entidad).

Tiene métodos para obtener:

1. obtener el número de serie: `getNumSerie();`

2. el tiempo de creación: `getTiempoDeCreacion();`

3. la edad de la entidad `double getEdad();`

4. el tiempo en generar la próxima entidad en función de la distribución elegida para el retraso: `Retraso(String nombre)`, donde nombre es la denominación con la que se registró la extensión de `Entidad` .

Con estos métodos básicos se pueden administrar las entidades necesarias en el modelo simple. Si se precisaran entidades más elaboradas, se podría extender esta clase dotándola de los atributos y métodos necesarios.

En el modelo de Arena las entidades existen hasta que entran en los bloques “Dispose”. En Java no es necesario destruir los objetos de forma explicita, pues en el momento en que pierden las referencias pueden ser recogidos por el recolector de basura y ser destruidos liberando sus recursos [Frou05]. Sin embargo, por propia experiencia, se ha considerado conveniente destruir de forma explicita las entidades creadas, y caso de no ser destruidas de esta forma deben quedar referenciadas en un objeto, que sirva al diseñador, en la fase de construcción del modelo, para chequearlo y poder rastrear posibles errores de diseño. Con este fin, se ha creado una tabla hash [Lewi06] dentro de Entidad: `map`, en forma de tabla estática (es decir, común para todas las entidades). Cada vez que se crea una entidad se carga su referencia en dicha tabla usando como clave su número de serie, cada vez que se destruye una entidad (paso equivalente al bloque dispose de Arena) se retira dicha entidad de la tabla mediante el método `Destruye()`. Existe la posibilidad de borrar todas las entidades de la tabla, creando una nueva tabla vacía, mediante `DestruyeTodos()` (esto no asegura que desaparezcan si han quedado referenciadas en otros elementos del modelo como las colas). Mantener las entidades en una tabla, si no se destruyen,

permite consultar dicha tabla para intentar ver las características de las que han permanecido e intentar ver los fallos en el modelo.

Como cada entidad presente consume recursos, un error que se podría presentar es que se mantuviera un número de entidades vivas que sobreocuparán los recursos del equipo. Al crear una nueva entidad se chequea el número de entidades presentes en la tabla y si se supera el valor asignado a la variable `limiteEntidadesVivas`, se detiene la simulación dando un aviso del problema. Si la causa es que quedan en la tabla y no se quiere usar el método para destruirlas, se pueden crear en lugar de con el constructor `Entidad()`, que las referencia en la tabla con el constructor alternativo `Entidad(false)`, que no las incluye en la tabla (map), por lo que no es necesario destruirlas (ni posible mediante `Destruye()`, ni tampoco tendremos un lugar en el que intentar encontrarlas para chequear problemas). Esta alternativa sólo es práctica si el modelo está bien chequeado y los parámetros de configuración son correctos, ya que si el modelo tiene fallos o si los parámetros fijados para la simulación llevan a valores grandes de entidades presentes, tampoco el sistema tendrá manera de evitar quedarse sin recursos, por lo que el autor no recomienda usar dicho constructor.

El concepto de "recurso" de Arena y las colas (Queues) que se organizan en torno a él tampoco se implementa en las librerías. El diseñador debería usar los objetos Java que considerara oportunos para su gestión. No obstante, para el modelo simple se ha creado la clase `simulacion/Recurso.java`. Para cada recurso diferente que sea necesario en el modelo se crea una clase que extienda a `Recurso.java`. En nuestro ejemplo se han creado dos: `simulación/Empaquetador.java` y `simulación/Reparador.java`. La clase `Recurso.java` contiene dos listas [Lewi06] :

1. `listaOcupados`, que incluye los tiempos en que un recurso inició su ocupación y todavía no ha sido liberado. Tiene utilidad para calcular el tiempo total de ocupación del recurso en el transcurso de la simulación.

2. `colaEspera` en la que se incluyen las entidades que esperan disponer del recurso junto con el instante de tiempo en el que comenzaron a solicitarlo. Es decir gestiona la cola asociada a dicho recurso.

Los métodos públicos que pone a disposición del diseñador para gestionar los recursos son los siguientes:

1. `public void PonEnCola(Object entidad)` añade a la cola del recurso la entidad pasada junto con el valor de tiempo de simulación en que se hace la solicitud.

2. `public boolean HayEnCola()` devuelve `true` si hay entidades esperando en la cola y `false` en caso contrario.

3. `public int NumEnCola()` devuelve el número de entidades en la cola del recurso en el momento en que se invoca.

4. `Object AtiendePrimero()` si hay entidades en la cola del recurso se retira devuelve la primera y actualiza en Recursos las variables necesarias para calcular estadísticas.

5. `public double Ocupa()` devuelve el tiempo de ocupacion, de acuerdo con la distribución aleatoria de tiempos de ocupación, de una unidad de recurso al tiempo que la retira de disponibles y actualiza las variables estadísticas, si no hay recursos devuelve `NO_DISPONIBLE`, (-1).

6. `public void Inicializa(int recursosInciales,RandomVariate distribucionServicio)` y `public void Inicializa(int recursosInciales)` inicializan los parámetros del recurso. El primer método actualiza la distribución aleatoria de tiempos de ocupación del recurso, el número de unidades iniciales disponibles y el número de operaciones finalizadas por el recurso. El segundo solamente el número de recursos iniciales y operaciones finalizadas.

7. `public void Libera()` libera una unidad del recurso y actualiza las variables estadísticas.

8. `public int ObtenMaxOcupados()` devuelve el máximo de unidades de recurso ocupadas en algún momento.

9. `public double ObtenOcupacion()` devuelve el la fracción del tiempo total de tiempo disponible por una unidad de recurso que ha sido ocupada. Se obtiene dividiendo el tiempo total de ocupación de todas las unidades por el tiempo total de simulación y por el número total de unidades de recurso puestas en juego.

10. `public int ObtenMaxEnCola()` devuelve el número máximo de entidades que han esperado en algún momento en la cola.

11. `public double ObtenTiempMaxEnCola()` devuelve el tiempo máximo que una entidad ha esperado en algún momento en la cola.

12. `public double ObtenTiempMedEnCola()` devuelve el tiempo medio que han esperado las entidades en la cola.

13. `public int ObtenMaxDisponibles()` devuelve el número máximo de unidades de recurso que se pusieron a disposición.

14. `public int ObtenDisponibles()` devuelve el número de entidades disponibles en el momento en que se invoca.

15. `public long ObtenOperacionesConcluidas()` devuelve el número total de operaciones realizadas por los elementos del recurso a lo largo de las simulación.

El recurso concreto a representar requerirá de nuevos métodos y posiblemente sobrescribir alguno de los proporcionados. Los dos usados en el modelo simple, `simulacion/Empaquetador` y `simulación/Reparador,` sólo han requerido que se sobrescribiera el método creador de la clase. Es decir, se han creado dos recursos diferentes sin apenas incluir código.

Normalmente, el modelo creado se ejecutará para distintos juegos de parámetros con el fin de encontrar los que proporcionan un comportamiento óptimo. Para poder variar de forma sencilla los parámetros de ejecución, se sugiere que lo más conveniente es crear variables publicas de instancia para la clase que extiende `SimEntityBase` (la clase que realiza la simulación), en las que guardarían dichos parámetros, los métodos `doXXX(...)` tomarían su valor para la ejecución de dichas variables. Su valor se puede actualizar, si la clase cuenta con un método `main()` a través de los parámetros de llamada. O bien si se dota de una GUI, la actualización podría ser mucho más flexible a partir de los menús suministrados por ella. A continuación, se indica a grandes rasgos cómo se ha creado la sencilla GUI del modelo simple.

A los eventos necesarios para la simulación propiamente dicha, se pueden añadir eventos que tiene como utilidad la posibilidad de obtener y guardar datos de la simulación propiamente dicha. En este modelo sencillo se ha creado un evento de este tipo, `doFinal` que en la Figura 2.11 se

ha representado en rojo, en la GUI del modelo simple se encarga de generar el mensaje con la información final de la simulación, de presentarla en el campo de texto y llevar la barra de progreso al 100% de la simulación.

## 2.9.-Diseño de la interfaz gráfica de usuario para el modelo simple

La clase `Nucleo` que extiende `SimEntityBase` para realizar el trabajo de la simulación y puede ser ejecutada mediante el método `main(...)`. Esto es lo más sencillo pues sólo se precisaría para modelo simple de las clases del paquete `simulacion`. Pero esto la limitaría en cuanto a la posibilidad de configuración e interacción con el usuario, por lo que es más conveniente llamarla desde una GUI que asista al usuario en la modificación de parámetros de simulación, permita visualizar resultados obtenidos durante y al final de la simulacion, y la posibilidad de guardar o no dichos resultados. Para integrar la clase `Nucleo` con la GUI, se lanza `Nucleo` como tarea desde la clase principal de la GUI (`ventanas/BarraMenus`), para lo cual `Nucleo` debe implementar la interfaz `Runnable`, que añade el método `run` (no confundir con `doRun`) en el cual se realiza la tarea de lanzar la simulación, cosa encomendada a `main(..)` si se prescinde de la GUI. En el modelo simple se puede optar por las dos opciones (usando la GUI, se ofrecen posibilidades adicionales como ya se ha dicho).

Para que la GUI interaccione con la clase `Nucleo`, la clase principal de la GUI, `ventanas/BarraMenus` crea la instancia de `Nucleo nucleo`, usando dicha instancia los conponentes de la GUI pueden hacer referencia a componentes de `Nucleo` bien para obtener sus valores y presentarlos al usuario, bien para cambiar sus valores configurar de esa manera la simulación. La GUI consta de 6 tipos de elementos: una barra de menús con sus menús y menuitems, tablas gráficas desde las que se pueden presentar valores al usuario, diálogos para

abrir o guardar en ficheros, tablas gráficas desde las que se pueden introducir valores, un cuadro de texto para presentar resultados y una barra de progreso que nos indica el avance de la simulación. Para facilitar la programación de la GUI, se han creado dos clases abstractas `ModeloPanelDeDatos` y `ModeloPanelResultados`.

`ModeloPanelResultados` presenta un tabla gráfica dentro de un `JScrollPane` (que según el tamaño aporta barras de desplazamiento), la creación de un objeto que estienda esta clase precisa tan sólo que se invoque el creador "`ModeloPanelResultados(String[] nombresFila, String[] nombresCol)`, pasando los nombres de las filas y de las columnas de las celdas que recogerán los valores de los parámetros a visualizar e implementar el método abstracto:

```
public abstract void Actua(PropertyChangeEvent evt)
```

En este método se debe alterar el valor de las celdas de la tabla en función de las propiedades de `evt`: `evt.getPropertyName()` y `evt.getNewValue()` que recogen el nombre de la propiedad que cambia y el valor del cambio. `PropertyChangeEvent` es lanzado por `Nucleo`, la clase extensión de `ModeloPanelResultados` debe ser registrada como `listener` de dicha propiedad de `Nucleo`. El registro como `listener` se ha hecho poco después de instanciar la clase que extiende `ModeloPanelResultados` en `BarraMenus` usando la variable `nucleo` que nos permite referenciar parámetros de la clase `Nucleo`. En `Nucleo`, cuando se produzcan cambios de la propiedad que deban ser presentados, se lanza un `PropertyChangeEvent` mediante `firePropertyChange("nombre", nuevo valor)`. A continuación, se ilustra el proceso para `PanelResultados` que es la única extensión de la clase abstracta realizado en el modelo simple.

Primero implementamos el método abstracto:

```
public void Actua(PropertyChangeEvent evt) {
              String propiedad = evt.getPropertyName().toString();
```

```
            try {
                    Long valor = (Long) (evt.getNewValue());
                    if ("generados".equals(propiedad)){
                            this.daValor(valor, 0, 0);
                    }
                    else if ("correctos".equals(propiedad)){
                            this.daValor(valor,1, 0);
                    }
                    else if ("defectuosos".equals(propiedad)){
                            this.daValor(valor, 2, 0);
                    }
                    else if ("empaquetados".equals(propiedad)){
                            this.daValor(valor, 0, 1);
                    }
            } catch (Exception e) {
                    System.out.println(e.toString());
            }

    }
```

Como se puede ver en el código se van a presentar en la tabla los valores de `generados`, `correctos`, `defectuosos` y `empaquetados` y las posiciones de la tabla que ocuparán, utilizando el método `daValor(valor, fila, col)` de la clase abstracta que recibe como parámetros el valor a representar y los índices de la celda en la que se representará. como se puede ver en el código no todas las celdas van a contener valores (para las celdas [1,1] y [2,1] no se producen modificaciones) en el ejemplo.

El siguiente paso es crear un objeto de la extensión de la clase en la clase principal de la GUI. En nuestro caso en “BarraMenus”, se ha optado por crear un array de paneles de resultados, por si se deseara crear más de uno. En el ejemplo sólo se creó un panel, es decir el elemento [0], con una tabla de dos filas y dos columnas, cuyos títulos se pasan.

```
ModeloPanelResultados[] panelesResultados;
.............
panelesResultados = new PanelResultados[1];
            panelesResultados[0] = new PanelResultados(new String[] { "generados",
                            "correctos", "defectuosos" }, new String[] { "elementos",
                            "empaquetados" });
```

A continuación, se declara como `listener` de las propiedades a representar la instancia creada:

```
((Nucleo) this.nucleo).addPropertyChangeListener("generados",
                                this.panelesResultados[0]);
```

.....

En el código se declara `this.panelesResultados[0]` como listener de los cambios en la propiedad `generados` de `nucleo`. Lo mismo se hace para las otras propiedades en las sentencias que siguen (no listadas aquí).

Finalmente, sólo queda lanzar la actualización correspondiente en la clase de `Nucleo` dentro del evento adecuado. En este caso, cada vez que se genere un nuevo componente, cosa que ocurre dentro del código de `doCreaComponente()`:

```
....
en = new Componente();
this.componentesCreados++;
firePropertyChange("generados", new Long(this.componentesCreados));
....
```

La clase abstracta permite representar los cambios de valor de la propiedad de forma simultánea a la simulación, permitiendo ver y comparar las variaciones de los valores en las distintas celdas mientras la simulación tiene lugar. Si no se desea ver la variación mientras ocurre la simulación (ejecución “silenciosa”), se pueden desactivar estas tablas, bien todas o una determinada utilizando los métodos `DesactivaTodos()` o `Desactiva()` respectivamente. Se pueden volver a activar utilizando `LiberaDesactivacion()`, que permite que se activen todos, y `Activa()`, que activaría un panel concreto.

`ModeloPanelDeDatos` es una clase abstracta que permite generar de forma sencilla una tabla en la cual se pueden editar los valores, una casilla de verificación que permite seleccionar la fila

de la tabla cuyos valores entrarán en el modelo y dos botones que permiten introducir los datos o cancelar la operación.

```
ModeloPanelDeDatos(Object destino, String[] filas,
                   String[] columnas, Object[][] datosIniciales)
```

El constructor de las extensiones de esta clase reciben la clase destino, en este caso la instancia de `Nucleo` (o `BarraMenus` ya que tiene referenciada la instancia `nucleo` de `Nucleo` que usamos), los nombres de las filas, los nombres de las columnas y el contenido inicial de las celdas, que suele ser conveniente que se busque en los valores por defecto de las propiedades a actualizar.

Es conveniente que las propiedades de `Nucleo` a actualizar sea declaradas como variables de instancia de la clase `nucleo`, de forma que con la referencia a `nucleo` que se pasa sean fácilmente accesibles al método abstracto (que se debe concretar en las extensiones de esta clase):

```
abstract void DevuelveValores();
```

Este método se ejecuta al pulsar el botón aceptar y sólo puede producir actualizaciones con


**Fig. 2.12**: Panel de Datos para actualizar recursos.

sentido en las filas de datos marcadas en su casilla de verificación ya que las no marcadas se devuelven a `null` por el método `ObtenModificaciones()`. A continuación, se muestra y comenta la implementación de este método para la extensión de la clase abstracta: `LeeRecursos`, esta clase presenta un panel con el aspecto mostrado en la Figura 2.12.

Consta de 6 celdas. La primera columna, con los nombres de fila, no es editable. La última columna con la casilla de verificación de fila bajo el nombre `actualizar`, contiene las casillas de verificación que marcan las filas con valores a introducir en el modelo. La casilla de verificación de la primera fila es especial, ya que activa o desactiva a las demás casillas. La mayor utilidad de ese comportamiento se daría en paneles con muchas filas, pues permitiría marcar o desmarcar todas las filas desde la primera. Las demás casillas de verificación sólo actúan sobre ellas mismas. El botón `Aplicar` permite introducir los parámetros chequeados en el modelo y el botón cancelar permite dejar la ventana sin introducir cambios. El creador de la ventana se invoca con el código siguiente:

```
new LeeRecursos(dest, new String[] { "empaqutadores", "reparadores" },
        new String[] { "número" },new Integer[][] { { new Integer(
((Nucleo) dest.nucleo).nEmpaquetadores) },{ new Integer(((Nucleo) dest.nucleo).nReparadores) }
});
```

Se pasan los títulos de filas, columnas y enlace con las variables de `Nucleo` para la lectura de los valores iniciales, en `dest`, se pasa una referencia a `BarraMenus` que permiteacceder a `Nucleo` a través de la variable `dest.nucleo`. El resto consiste en implementar el método abstracto, como se muestra a continuación:

```
public void DevuelveValores() {
                Object[][] res = this.ObtenModificaciones();
                Nucleo nuc = (Nucleo) ((BarraMenus) this.destinoDatos).nucleo;
                if (res[0] != null)// cambiamos empaquetadores
                        nuc.nEmpaquetadores = ((Integer) res[0][0]).intValue();
                if (res[1] != null)// cambiamos reparadores
                        nuc.nReparadores = ((Integer) res[1][0]).intValue();
        }
```

El método `ObtenModificaciones()` devuelve un `array[][]` de objetos que se guarda en `res`. Si la casilla de verificación no se marca dicha fila de `res[..]` vale `null`, por eso se comprueba

si no tiene ese valor antes de lanzar las actualizaciones en los parámetros de Nucleo. Como se ve, se crea con poco código ventanas que actualicen parámetros de Nucleo.

En una aplicación compleja introducir los parámetros de forma manual, si estos son muchos no es una forma adecuada, ya que para luego analizar resultados es necesario guardar constancia de los parámetros utilizados y los resultados obtenidos. Se ha dotado a este modelo sencillo de la posibilidad de leer parámetros desde un archivo de texto plano, mediante métodos de la clase accesorios/AnalPauta. Concretamente, su método:

```
public String[] ObtenDato(String texto, String marca1, String marca2,  String marca3)
```

Con texto se introduce el String a analizar, marca1, marca2 y marca3 son cadenas que delimitan el nombre del parámetro a introducir y su valor. Por ejemplo, si se toma marca1="<<" marca2= ">> " y marca3=">> "; en el texto "<<nEmpaquetadores>>4>> " se leería un valor para nEmpaquetadores de 4. Se podrían haber utilizado clases Java estándar como ficheros de properties, pero este método permite más flexibilidad para introducir valores para variables multidimensionales que deban ocupar varias líneas, y da facilidad para introducir comentarios.

Para seleccionar un fichero como entrada o salida se puede usar la clase ventanas/SelectorFicheros que muestra un dialogo de gestión de archivos y la selección se devuelve como un File o un String que apunta al destino del cual leer o guardar mediante los métodos Obten() u ObtenString() respectivamente. La carga se hace con el método cargar(String fich) de accesorios/AnalPauta que devuelve un String con el contenido del fichero, y guardar fichero se hace con el método Guarda(String fich, String ValModerno) de accesorios/Guardar, que carga fich con el texto ValModerno.

## 2.10.-Estadísticas en Simkit

Para mostrar algunas posibilidades de Simkit, se ha añadido la generación de cálculos estadísticos usando la clase `simkit.stat.MultipleSimpleStatsTally`, del paquete `simkit.stat`, que permite obtener estadísticas de un array de parámetros. Nosotros la cargaremos con el número de empaquetadores que hay libres cuando se produzca un cambio en su número en la posición 0 del array, con el número de componentes en la cola de empaquetadores en la posición 1 y con el número de componentes en la cola de reparadores en la posición 2. Para su uso se crea una instancia asignándole un `String`, con lo que si fuese necesario se podrían crear distintas instancias:

```
MultipleSimpleStatsTally  iss = new MultipleSimpleStatsTally("miEstad");
```

Luego se puede resetear para borrar cualquier historia anterior: `iss.reset()`.

Finalmente, cada vez que se de la circunstancia cuya estadística se quiere obtener se carga el valor a muestrear. En nuestro caso cuando cambie el número de empaquetadores, o elementos en las colas.

Es decir, en los eventos `doCorrecto()` y `doEmpaquetado(Entidad en)`en los cuales puede caer o aumentar el número de empaquetadores, dentro del código se ejecuta el método:

```
iss.newObservation(emp.Disponibles(),0);
```

o bien en `doCreaComponente()` donde se colocan en cola de empaquetado si es correcto se invoca `iss.newObservation(emp.NumEnCola(), 1)` y si es erróneo se pone en cola del reparador lugo la estadística se carga con `iss.newObservation(rep.NumEnCola(), 2)`.

Ya sólo queda, cuando se haya recabado la información, pedir los valores estadísticos que esta clase calcula. Cosa que se hace en `doFinal()` donde se obtienen y se muestran de `iss` los siguientes parámetros estadísticos [Box 78]:

máximo: `iss.getMaxObs(pos);`

media: `iss.getMean(pos);`

desviación standard: `iss.getStandardDeviation(pos);`

varianza: `iss.getVariance(pos);`

donde pos sería 0, 1 o 2 según se quieran estadísticas de empaquetadores libres, componentes en cola de empaquetador o componentes en cola de reparador respectivamente.

## 2.11.-Variables aleatorias definidas por el usuario

Otra posibilidad es la de generar nuestras propias variables estadísticas [Papo91]. Simkit es capaz de generar las distribuciones más frecuentes, como podemos comprobar dentro del paquete `simkit.random`. Si pese a todo no encontramos la que necesitamos, no sólo podemos crearla con nuestros requisitos sino que podemos incluirla dentro de `RandomVariateFactory`, que es una factoría que genera instancias de las distribuciones del paquete estadístico de Simkit mediante su nombre, o de paquetes creados por el usuario siempre que sigan el modelo que tiene previsto Simkit. Para que Simkit incluya las distribuciones de nuestro paquete estadístico como clases instanciables desde `RandomVariateFactory` es necesario que extiendan `RandomVariateBase`. La clase abstracta `RandomVariateBase` obliga a implementar los métodos:

1. `public double generate()`: que devuelve el valor aleatorio que genera nuestra distribución;

2. `public void setParameters(Object[] arg0)`: que asigna los parámetros de configuración;

3. `public Object[] getParameters()`: que permite obtener los parámetros con que se configuró.

Como fue necesario implementar una variable aleatoria que genere un retraso exponencial con parámetro variable de forma periódica, hemos adaptado nuestra variable para que pueda ser suministrada desde `RandomVariateFactory`. Para lo cual hay que implementar los métodos abstractos ya comentados y los demás necesarios para el funcionamiento correcto de nuestra clase.

Dada la manera en que se obtienen los valores de nuestra distibucion cuando la usamos es conveniente crear tres versiones más del método `public double generate ()`:

```
public double generate(double t);
public double generate(double t, Double[] param)
public double generate(double t, double[] param)
```

El método `generate()` devuelve la variable para el tiempo de simulacióncorriente, `generate(t)` permite indicar el instante de tiempo para el que se quiere la variable; los otros dos además de indicar el tiempo permiten alterar los valores fijados de los parámetros periódicos (se fijan con el constructor o con el método `setParameters(Object[] arg0)`).

Los parámetros son números que indican el número de ocurrencias esperadas en el intervalo de tiempo unidad (en el modelo simple la unidad de tiempo es el minuto, por lo que si se pasan como parámetros 2, 3, 0, 5: se esperarían 2 ocurrencias en el primer minuto, 3 en el 2º, 0 en el 3º, 5 en el 4º y a partir del 5º minuto se repiten los números de ocurrencias de forma indefinida) este esquema se corresponde con el de Arena pero en la exponencial normal de Simkit el parámetro es el inverso de lo comentado. Si sólo se pasa un parámetro en nuestra variable, se convierte en una exponencial normal, pasar 10 en nuestra exponencial equivale a pasar 1/10 para

la clase que genera exponenciales de Simkit es decir 0.1 en `ExponentialVariate` del paquete `simkit.random`. Nuestra variable se ha creado con el nombre `ExpoParVarPer` del paquete `paqeteEstadistico`, para registrarla en `RandomVariateFactory` ejecutamos en el método creador de `Nucleo` la sentencia:

```
RandomVariateFactory.addSearchPackage(“paqueteEstadistico”);
```

en la que indicamos al método estático que busque variables estadísticas dentro del paquete pasado como `String`, a partir de ese momento podemos obtener una instancia de `ExpoParVarPer` con sólo invocar el método estático:

```
RandomVariateFactory.getInstance(“ExpoParVarPer” ,new Double[]{......}, semilla);
```

donde se pasa el nombre, los parámetros necesarios para configurarla y una semilla para generar los números pseudoaleatorios, tal y como se obtienen las distribuciones de `simkit.random`.

## 2.12.-Estructura de la aplicación modelo simple

La aplicación modelo simple se ha implementado con 4 paquetes:

1. `simulacion`; contiene las clases `Nucleo`, `Recurso`, `Empaquetador`, `Reparador`, `Entidad` y `Componente`. La clase fundamental es `Nucleo` que es la extensión de `SimEntityBase`, las que le acompañan son modelos y extensiones para asistir en el diseño de recursos y entidades. `Recurso` es el modelo para los recursos y se extiende en `Empaquetador` y `Reparador` para representar los usados en nuestro modelo. `Entidad` es la clase modelo para las entidades y se extiende en `Componente` para representar los

componentes de modelo simple. Si no se quiere GUI con sólo este paquete se pueden hacer las simulaciones.

2. ventanas; contiene las clases `BarraMenus`, `SelectorFicheros`, `ModeloPanelResultados`, `PanelResultados`, `ModeloPanelDeDatos`, `LeeRecursos`, `LeeTSimulacion`, `LeeDistEst`, `LeeNParam` y `LeeParam`. `BarraMenus` es la clase principal que contiene instancias de las demás y además lanza a `Nucleo` como tarea. `SelectorFicheros` presenta un diálogo de selección de ficheros. `ModeloPanelResultados` es el modelo de panel de resultados y `PanelResultados` una extensión que se incluye en `BarraMenus`. `ModeloPanelDeDatos` es la clase modelo de los paneles de datos y las clases `LeeXXX` son extensiones de ella que permiten leer diversos datos.

3. `accesorios`; contiene las clases `AnalPauta`, y `Guardar`. Es un paquete de utilidades, de la primera clase se usan dos métodos uno para leer el contenido de un archivo y otro que analiza el contenido para obtener datos de simulacion, la segunda permite guardar un texto en un archivo. Este paquete es necesario para que funcionen algunos submenús de la GUI.

4. `paqueteEstadistico`; con sólo la clase `ExpoParVarPer` esta clase permite generar una exponencial de parámetro variable. A la vez ha servido para ilustrar como se pueden integrar las distribuciones creadas por el usuario con las propias de `simkit.random`. Este paquete es necesario para algunas opciones de la GUI.

A continuación mostramos los diagrama de clases simplificados que ilustran las dependencias del modelo simple en la Figura 2.13 se muestran las clases del paquete simulación.

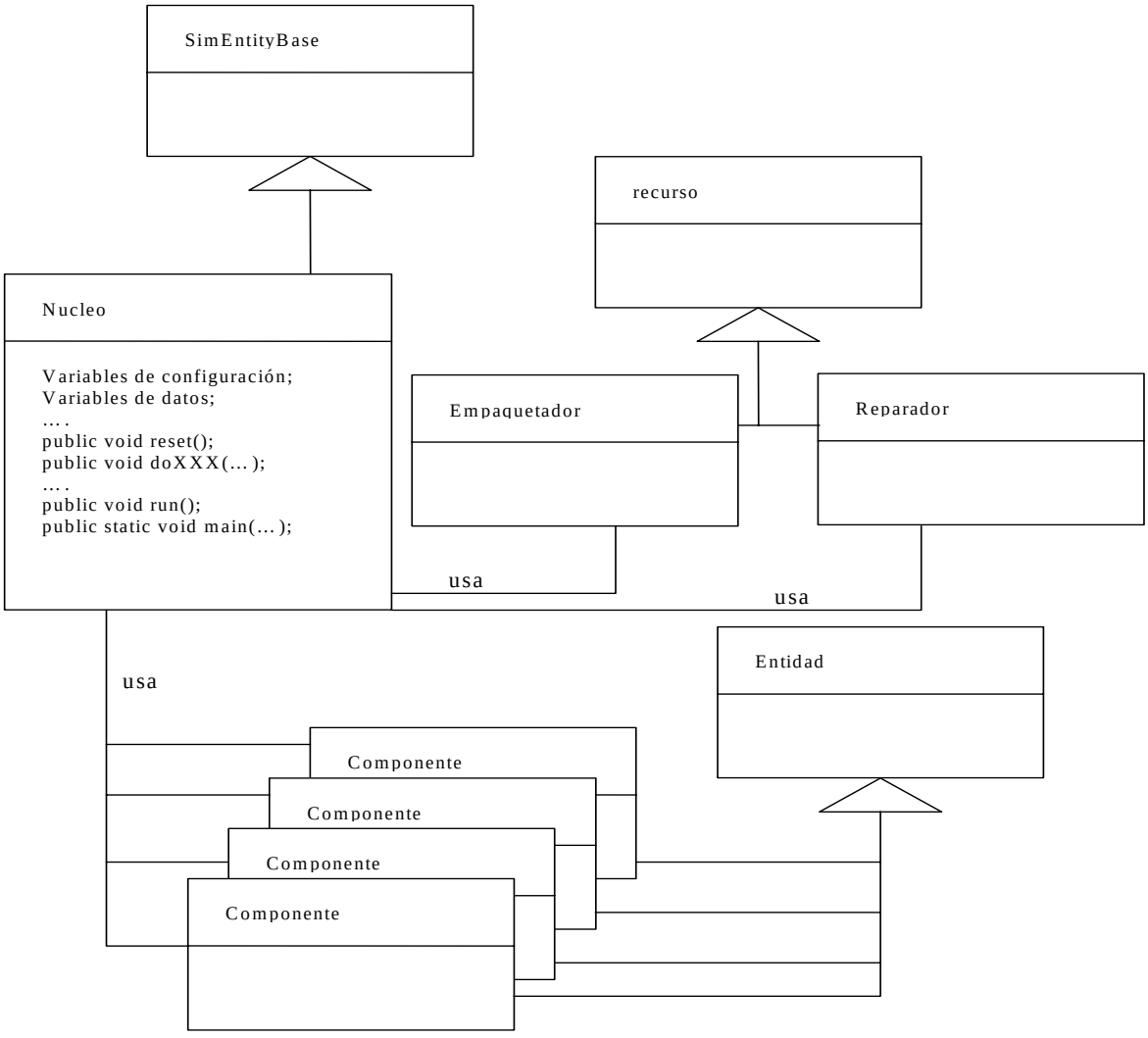


**Fig. 2.13**: Diagrama de clases simplificado del Modelo Simple.

En la Figura 2.14 se representa las relaciones de las clases de la GUI con `Nucleo`, que es la clase que realiza la simulación.

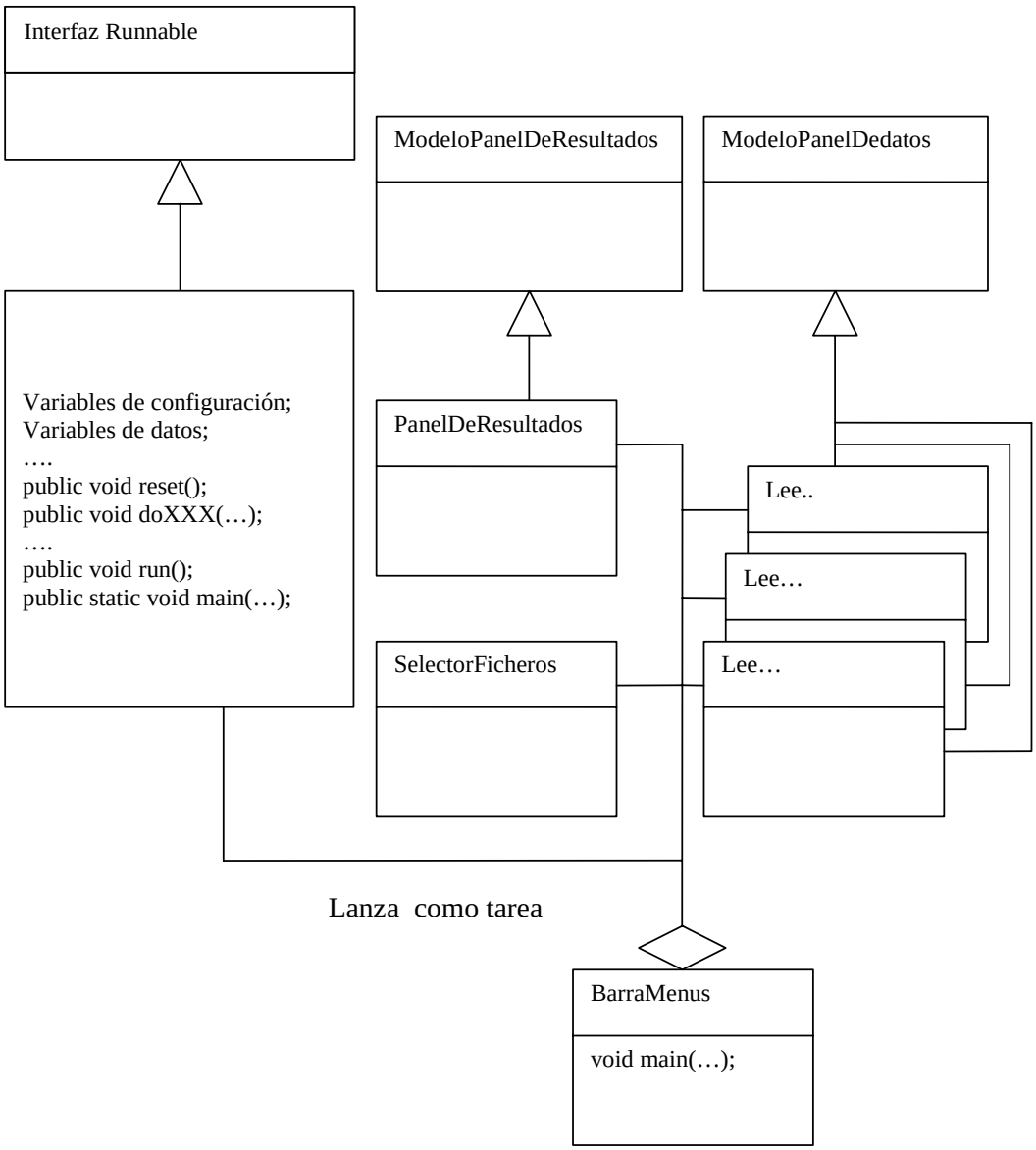


**Fig. 2.14**: Diagrama de clases simplificado de la GUI de ModeloSimple.

## 2.13.-Conclusiones

Se han descrito las fases necesarias a seguir para desarrollar un proceso de modelado.

Se han descrito de forma detallada las funciones básicas de las herramientas Arena y Simkit basadas en los paradigmas “orientado al proceso” y “orientado a eventos” respectivamente. Se han ilustrado las explicaciones desarrollando en paralelo en modelado de un caso sencillo (modelo simple). Se ha aprovechado dicho modelo para introducir objetos reutilizables de carácter general para complementar las librerías de Simkit como son las clases incluidas en el paquete `simulación`:

`Entidad.java` y `Recurso.java`;

o para facilitar el desarrollo de una GUI que facilite el control de las simulaciones en las clases incluidos en el paquete ventanas:

`BarraMenus.java`, `ModeloPanelDeDatos.java`, `ModeloPanelResultados.java` y `SelectorFicheros.java`;

Clases que facilitan tareas como lectura y guardado de datos en ficheros de texto, el análisis del contenido del los mismo para obtener información, se han incluido en el paquete accesorios en las clases:

`AnalPauta.java` y `Guardar.java`;

Una clase que ilustra como generar paquetes de distribuciones estadísticas para que se puedan ser incluidos como tales por la clase RandomVariateFactory del paquete random, se ha incluido en el paquete paqueteEstadistico la clase:

`ExpoParVarPer.java`.

# Capítulo 3.-Modelado de un establecimiento de comida rápida

## 3.1.-Introducción

Uno de los objetivos es demostrar que las librerías Simkit pueden ser útiles para desarrollar modelos de un nivel de complejidad que no puedan ser abordados por la versión académica de Arena. Con este fin se describe en detalle un modelo (simulación de un establecimiento de comida rápida) con un nivel de complejidad que cuando se realice la implementación en Arena se precise su versión completa. En este Capítulo se describen los objetos que constituirán el modelo, las leyes que rigen su comportamiento, las interacciones entre ellos, sus parámetros de configuración, las entradas aleatorias etc. Estos elementos deberán traducirse a objetos de los dos entornos Arena y Simkit y comparar los resultados en ambos entornos, pero eso es tarea de capítulos posteriores.

## 3.2.-Objetivo de la simulación

Se modela un establecimiento de comida rápida con el fin de identificar los parámetros y valores de los mismos que conducirían a una optimización del rendimiento de dicho establecimiento. Se van a utilizar como herramientas: Arena 7.0 y una máquina virtual Java usando las librerías estándar y las específicas de Simkit. Ambos entornos permiten diseñar y ejecutar en un PC con procesador pentium (III) o superior con Windows XP como sistema operativo.

## 3.3.-Descripción del funcionamiento del sistema

Definición del sistema y formulación del modelo. Los elementos que intervienen en un establecimiento real se han simulado mediante las siguientes simplificaciones:

1. Clientes: Se ha supuesto que su llegada sigue una distribución exponencial cuyos parámetros varían en función de las horas del día, de la calidad del servicio y del precio medio de los productos del establecimiento. Se consideran que llegan en grupos de personas que pueden variar desde 1 hasta 10, con sus probabilidades que decrecen con el número de forma que para simplificar se ha despreciado los grupos de más de 10 personas. (Ver Tabla 3.2 al final del capítulo).

2. Cajeros: simulan al personal del establecimiento con ese cometido. Se trata como un recurso planificable horariamente, sin embargo en la práctica en un caso real, (y en las simulaciones realizadas la libertad de planificación se vería restringida por criterios laborales). Su distribución diaria en las pruebas se muestra en la Tabla 3.3 al final del capítulo.

3. Personal de cocina: simulan al personal encargado de generar los productos del establecimiento. Si bien se ha considerado solamente una sola clase (cocineros) de personal de cocina en las simulaciones realizadas, el modelo elaborado admite hasta tres tipos de personal diferente. Al igual que los cajeros son planificables cada hora aunque se han tenido también en cuenta para la simulación ligaduras que impondrían las condiciones laborales usuales(8 horas diarias por persona y 5 días de trabajo a la semana). Las distribuciones ensayadas se muestran en la Tabla 3.1 al final del capítulo.

4. Mesas: para su simulación se han considerado dos tipos de recursos, ‘mesas’ y ‘mesas2’, sus características son:

   ‘mesas2’ son mesas fijas que permiten que se acomoden hasta 4 clientes;

   ‘mesas’ son mesas que se pueden mover. Si están aisladas se pueden ocupar hasta por 4 clientes. Si se mueven para hacer agrupaciones se dan las siguientes situaciones: si se juntan 2 pueden ser ocupadas hasta por 6 clientes; si se juntan 3 hasta por 8 clientes y finalmente si se juntan 4 puede ser ocupada la agrupación hasta por 10 clientes. No se agrupan más de 4 ya que el grupo mayor de clientes es de 10 y el interés de agruparlas es proporcionar un espacio común en el que acomodar al grupo de clientes.

5. Productos del restaurante, se considera que se generan tres tipos de menús diferentes, se han denominado “pizza”, “ensalada” y “hamburguesa”, cada uno con sus propiedades fijas tales como las cantidades de materias primas para su elaboración, la distribución de tiempo para su elaboración (distribuciones triangulares). Propiedades variables como probabilidad de elección de cada uno por los clientes que fluctúa de acuerdo con los precios relativos de los productos, (por ejemplo si aumenta el precio de la pizza su probabilidad de de elección cae frente a los otros dos productos cosa bien conocida en economía [Sier82]. Para ver como es dicha variación ver Ec (3.4). Otra propiedad es el precio de cada producto y que se variará como parámetro para lograr el objetivo del establecimiento que es generar el máximo beneficio.

6. Materias primas para elaborar los productos, se ha considerado que se necesitan 13 productos para generar los menús. A cada uno de ellos se le ha asignado un nombre. Cada materia prima entra en una cantidad fijada en cada menú (en algún caso puede ser 0). Cada materia tiene un precio, que no varía en el transcurso de una simulación. Otras

propiedades que no varían en una simulación serían: la cantidad que se adquiere por pedido, el tiempo de caducidad, el nivel de existencias que desencadena un nuevo pedido (se fija un umbral de existencias, si se consume de forma que se baje de ese nivel se realiza pedido), el tiempo que falta para que las existencias caduquen que desencadena un nuevo pedido (se fija un porcentaje del tiempo de caducidad de una materia, si en la simulación se sobrepasa ese tiempo, se está en riesgo de quedarse sin materia por caducar, por lo que se debe hacer pedido), el precio fijo a pagar por pedido.

## 3.4.-Modelado del proceso de llegada de clientes

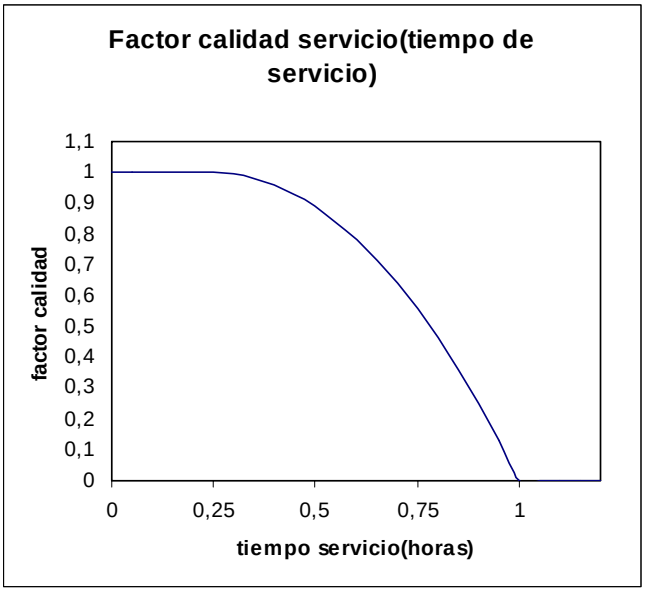


**Fig. 3.1**: Factor calidad frente a tiempo medio de servicio.

Los conceptos descritos en el apartado anterior interaccionan de acuerdo a las siguientes reglas o comportamientos que simularían facetas de un caso real.

Los clientes llegan al establecimiento en cantidades que fluctúan a lo largo de las horas del día, se han establecido unos valores máximos horarios de afluencia para un nivel óptimo de

precio y de calidad de asistencia, como parámetros externos (véase Tabla 3.1 al final del capítulo). El nivel de afluencia real al establecimiento se obtiene como una modulación del valor anterior multiplicando por un factor precio y por un factor calidad.

$$concurrencia = concurrencia_{optima} \times Factor_{precio} \times Factor_{calidad\ servicio} \tag{3.1}$$

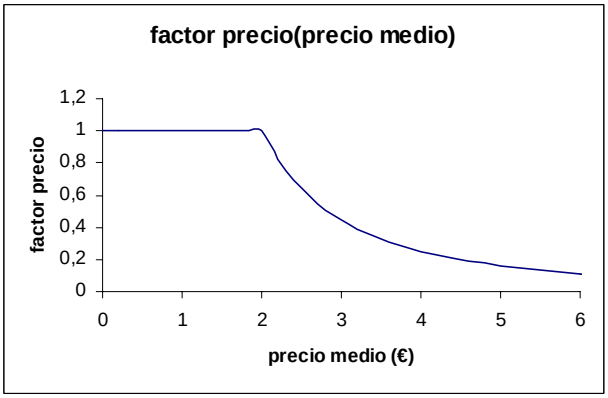


**Fig. 3.2**: Gráfica factor precio frente a precio medio.

Los factores precio y calidad dependen del precio medio y del tiempo medio de servicio respectivamente. Se muestran a en las Figuras 3.2 y 3.1 las gráficas de dicha dependencia.

El factor precio se obtiene a partir de los precios fijados para los menús (que no se cambian en una simulación), por lo que una vez iniciada ésta es un valor fijo.

$$Factor_{precio} = \begin{cases} 1\, si\ \ precio\ medio \le 2 Euros \\ \dfrac{4}{p_{medio}^2} si\ \ precio\ medio > 2 Euros \end{cases} \tag{3.2}$$

El factor calidad se obtiene en función de la calidad del servicio, y se mide esta mediante el tiempo medio de espera desde la llegada al establecimiento hasta obtener el menú para el consumo.

$$Factor_{calidad\ servicio}\begin{cases} 1\ si\ tiempo\ medio \le 0'25 horas \\ \frac{8}{9}+\frac{8}{9}t-\frac{16}{9}t^2\ si\ 0'25 \le tiempo\ medio \le 1 hora \\ 0\ si\ tiempo\ medio > 1 hora \end{cases} \quad (3.3)$$

como este parámetro variará a lo largo de la simulación, su influencia determina que la afluencia de clientes oscile a lo largo de cada simulación. Se supone que los valores para los parámetros que determinan las tres influencias se obtendrán mediante estudios de mercado, una vez obtenidos serían constantes de la simulación. En el modelo desarrollado se ha obtenido unos valores tabulados que corresponderían a la concurrencia horaria óptima en las mejores condiciones de precio y calidad de servicio, modelándose las dos últimas influencias mediante funciones con valor máximo de 1 que actúan como factores de la concurrencia optima. De forma que la concurrencia efectiva en un momento de la simulación se obtiene como se indica en Ec. (3.1).

El factor precio, se ha considera que sigue la relación demanda precio usual en economía [Sier82], según suben los precios cae la demanda. Se considera que para un precio medio bajo que se ha tomado de 2€ se alcanza la concurrencia óptima y mayores rebajas no aumentan la demanda, es decir que para precios menores de 2€ el factor precio vale la unidad. Para precios mayores sigue una curva típica demanda /precio. Se ha elegido una con elasticidad

mayor que la unidad que es adecuado para productos que no son de primera necesidad [Sier82]. Concretamente la función con las que se han realizado las simulaciones es la de la Ec (3.2).

La influencia del tiempo medio para obtener los menús influye en el valor del factor calidad de servicio de acuerdo con la Ec. (3.3 ).

Los dos factores hacen que la demanda caiga rápidamente al aumentar precio medio y tiempo medio en obtener el menú ver las Figuras 3.2 y 3.1.

Tras generarse el grupo de clientes se decide el número de personas que lo forman de acuerdo con la distribución de probabilidad que se habría obtenido de estudios de mercado, (ver la Tabla 3.2 al final del capítulo).

## 3.5.-Modelado de ocupación de las mesas

Llegan al establecimiento pasando a la cola de entrada para obtener mesa. Se ha establecido que si hay 5 grupos esperando en la misma el grupo recién llegado desiste de intentar entrar y se retira. Si hay menos de 5 grupos se sitúa en la cola y si se sobrepasan los tiempos de espera fijados mediante una distribución triangular cuyos parámetros derivarían de estudios de mercado desistirían de la misma. Los grupos de clientes de menos de 5 personas pueden ocupar bien mesas fijas “mesas2” o bien las configurables “mesas”, mientras que los grupos de 5 o más sólo pueden optar a las segundas, ya que necesitan agruparlas para sentarse juntos. En la cola los grupos pequeños rebasan a los numerosos si hay libres mesas fijas, respetando la cola si el recurso libre es la mesa configurable. El número de mesas configurables que ocupa un grupo depende del número de personas que lo componen , pudiéndose obtener este número mediante la fórmula.

$$n_{mesa} = \left\lfloor \frac{num_{pers} - 0'25}{2} - 1 \right\rfloor + 1 \qquad (3.4)$$

(⌊...⌋ representa la parte entera de la expresión decimal del interior)

Los grupos que puede ocupar mesas fijas ocupan 1 cada vez (también se podía aplicar la fórmula anterior pero da valor 1 para los grupos de menos de 5 personas).

El tiempo que invierte cada cliente se empieza a contar a partir del instante de generarse el grupo y ponerse en la cola de entrada. No se promedian los tiempos de los clientes que no llegan a entrar por desistir.

## 3.6.-Modelado de la petición de comida

Tras obtener las mesas necesarias el grupo va a la cola del cajero para realizar el pedido, no se considera la posibilidad de desistir en esta cola, ni por tamaño de la cola, ni por el tiempo de espera. De todas las maneras con los parámetros externos que se han usado, no se dan tiempos de espera largos en la cola de los cajeros. El proceso de solicitud de menú se ha modelado con una espera triangular con una parte del tiempo fija y otra parte se ha hecho proporcional al número de personas que forman el grupo.

La petición del grupo se escinde en menús individuales de uno de los productos del establecimiento dicho producto se determina en función de los valores de la probabilidad que de acuerdo con los precios le corresponde.

Las probabilidades de un menú se determinan de acuerdo con la fórmula.

$$prob_i = \frac{prob_{0i} \times prec_{0i}}{prec_i \times \sum_{j=1}^{3} \frac{prob_{0j} \times prec_{0j}}{prec_j}} \qquad (3.5)$$

La fórmula anterior recoge el fenómeno corriente en economía que se sustituye un producto 1 por un producto 2 cuando sirven para la misma función y el primero incrementa su precio [Sier82]. También es una función de probabilidad siempre y cuando los valores de la ecuación sean positivos, cosa normal si se tiene en cuenta que representan lo siguiente:

Prob, son probabilidades y prec precios, los valores sub0 representan probabilidades y precios obtenidos en el estudio de mercado (probabilidades ideales y precios ideales), sus valores para las simulaciones están recogidos en la Tabla 3.3 al final del capítulo, con la formula anterior se calculan los valores a otros precios.

## 3.7.-Modelado de la preparación de la comida

A continuación cada menú pasa a iniciar su elaboración, el primer paso consiste en ver si hay cantidad suficiente de todos los ingredientes, y si los niveles de existencias y estado de caducidad de las materias primas recomiendan realizar un pedido. Se chequean todos los productos que intervienen en la generación del menú a elaborar. Si falta algún ingrediente no se puede elaborar el producto. Se ha establecido la posibilidad , tras un tiempo de espera (que simula el tiempo que se pierde en consultas para verificar que no se puede hacer el menú y su comunicación al cliente), de que bien se cambie el menú a otro o bien que se renuncie a consumir. Si el segundo menú fallara se permite elegir el tercero o bien desistir y si fallara el último se desistiría definitivamente, esto se regula mediante una espera triangular que modele el tiempo perdido y la probabilidad de desistir u optar entre los otros dos con probabilidades determinadas a partir

de las asignadas a los tres productos. En el caso de optar por tomar un segundo menú, las probabilidades derivan de las vigentes para los tres menús de acuerdo con la ecuación.

$$p_{2i} = \frac{p_{3j}}{p_{3i} + p_{3j}} \qquad (3.6)$$

si son los menús i y j los asequibles, las probabilidades que quedaría para i sería la expresada en la Ec. (3.6).

Para j, basta con cambiar i por j.

La Ec. (3.6) significa que si $P_{31}$=0'3, $p_{32}$=0'5 y $p_{33}$=0'2 y se agota el producto 2. Las probabilidades de escoger 1 y 3 cuando ya sólo hay dos menús disponibles se calcularían:

$p_{21}=p_{31}/(p_{31}+p_{33})=0'3/(0'3+0'2)=0'6$;

$p_{33}=p_{33}/(p_{31}+p_{33})=0'2/(0'3+0'2)=0'4$.

En la simulación, de la que se han recogido los datos, se han dado valores que obligan a desistir del consumo, y no se intentan segundos o terceros menús.

## 3.8.-Modelado de los stocks de alimentos

Las existencias de materias primas se mantienen mediante dos stocks, uno de productos de los que se están consumiendo y el segundo de productos en reserva. Cuando el stock en consumo se agota, se pasa el de reserva a consumo. Se ha fijado un nivel de existencias mínimo, que es configurable para cada producto, aunque en la simulación se ha fijado para todos en el 10% de la cantidad que aporta un pedido, si se baja de este nivel se genera un pedido. También, se controlan los tiempos de caducidad de las existencias. Cuando ha pasado el 90% del tiempo para caducar se genera pedido, (el valor 90% es configurable para cada producto, aunque es la cantidad que se ha fijado en la simulación para todos los productos). Si el producto llega a la caducidad se elimina lo que queda, cosa que ocasiona pérdidas. Cuando se dan las condiciones para realizar un pedido se chequea que este no esté ya en curso antes de realizarlo. Si cuando llega un pedido todavía queda producto en consumo, se almacena el pedido en reserva, hasta que se agote el producto en consumo.

Si no hay problemas de materias primas, se elaboran los menús. El tiempo que se invierte se modeliza mediante una espera triangular diferente para cada producto.

## 3.9.-Modelado del consumo

Tras su elaboración hay que reunir los pedidos para cada grupo (ya que cada componente del grupo hace su propio pedido y siguen un proceso de elaboración independiente). Una vez completada la petición de un grupo se pasa a éste para su consumo. El tiempo trascurrido desde la llegada del grupo al establecimiento hasta este momento, contribuye al tiempo medio global

que modula la asistencia al establecimiento pues es el parámetro temporal con el que se calcula el factor  calidad del servicio.

Los grupos pasan a consumir sus peticiones, se han establecido retardos distribuidos triangularmente, con parámetros que dependen de la composición de los menús del grupo y del número de comensales de que consta.

## 3.10.-Modelado de la preparación de la comida

Tras finalizar el grupo abandona el establecimiento.

## 3.11.-Modelado de las entradas aleatorias

Las entradas aleatorias se han modelado con las siguientes distribuciones:

1. La generación del tiempo de llegada de clientes se ha modelado distribuida exponencialmente, con fluctuaciones horarias que se repiten todos los días de acuerdo con los valores de la Tabla 3.1 al final del capítulo  (En Simkit se ha probado con un grupo todavía menos numeroso de cocineros, el grupo 3, (ver Tabla 7.9).

2. Los valores de los tipos de clientes se generan mediante una distribución discreta de probabilidad.  Los valores permitidos y sus probabilidades se listan en la Tabla 3.2 al final del capítulo.

3. Los valores de tiempo de espera en cola de entrada antes de desistir se distribuyen triangularmente con los valores recogidos en la Tabla 3.8 al final del capítulo.

4. El tipo de menú de cada cliente se generan mediante una distribución discreta cuyos valores de probabilidad se obtiene mediante la Ec. (3.5), que utiliza parámetros de la Tabla 3.3 al final del capítulo.

5. Los tiempos de fabricación de cada menú se distribuyen triangularmente de acuerdo con los valores de la Tabla 3.3 al final del capítulo.

6. Los tiempos de consumo, tanto para clientes solos como para clientes acompañados se distribuyen triangularmente con parámetros que se recogen también en la Tabla 3.3 al final del capítulo.

7. La probabilidad de probar con otro menú frente a no consumir caso de no haber ingredientes para elaborar el pedido inicialmente, será un parámetro cuyas probabilidades se fijan en la optimización, y tiene una distribución discreta.

8. La elección del menú de 2ª opción será mediante una distribución discreta en la que los valores serán los otros dos menús (el menú elegido inicialmente no se puede elaborar) con probabilidades derivadas de las de los dos mediante Ec (3.6).

9. Los tiempos de espera para obtener que se pase a elaboración el segundo o tercer menú (si no había ingredientes para los anteriores) se distribuyen triangularmente con parámetros recogidos en la Tabla 3.9 al final del capítulo.

TABLAS CON LOS PARÁMETROS DE ENTRADA PARA EL MODELO

| Tramo horario | llegada máxima de clientes | planificación de los cocineros | | | planificación de los cajeros |
|---|---|---|---|---|---|
| | | 0ª | 1ª | 2ª | |
| 00h-01h | 0 | 0 | 0 | 0 | 0 |
| 01h-02h | 0 | 0 | 0 | 0 | 0 |
| 02h-03h | 0 | 0 | 0 | 0 | 0 |
| 03h-04h | 0 | 0 | 0 | 0 | 0 |
| 04h-05h | 0 | 0 | 0 | 0 | 0 |
| 05h-06h | 0 | 0 | 0 | 0 | 0 |
| 06h-07h | 0 | 0 | 0 | 0 | 0 |
| 07h-08h | 0 | 0 | 0 | 0 | 0 |
| 08h-09h | 0 | 0 | 0 | 0 | 0 |
| 09h-10h | 0 | 0 | 0 | 0 | 0 |
| 10h-11h | 0 | 0 | 0 | 0 | 0 |
| 11h-12h | 120 | 10 | 10 | 5 | 1 |
| 12h-13h | 480 | 15 | 15 | 10 | 1 |
| 12h-13h | 750 | 35 | 35 | 30 | 1 |
| 13h-14h | 750 | 40 | 35 | 35 | 2 |
| 14h-15h | 360 | 15 | 15 | 10 | 1 |
| 15h-16h | 360 | 20 | 15 | 10 | 1 |
| 17h-18h | 750 | 35 | 30 | 30 | 2 |
| 18h-19h | 750 | 40 | 35 | 35 | 2 |
| 19h-20h | 600 | 35 | 30 | 25 | 1 |
| 20h-21h | 480 | 35 | 30 | 25 | 1 |
| 21h-22h | 360 | 20 | 15 | 10 | 1 |
| 22h-23h | 240 | 15 | 10 | 10 | 1 |
| 23h-24h | 0 | 5 | 5 | 5 | 1 |

**Tabla 3.1**: Planificación horaria de clientes, cajeros y tres opciones de cocineros.

| nº Personas | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| probabilidad | 0,2 | 0,24 | 0,12 | 0,15 | 0,07 | 0,1 | 0,03 | 0,04 | 0,03 | 0,02 |

**Tabla 3.2**: Probabilidades discretas de los distintos grupos de clientes.

| nombres | pizza | ensalada | hamburguesa |
|---|---|---|---|
| precios ideales | 5,75 | 4,25 | 6,75 |
| probabilidades ideales | 0,3 | 0,2 | 0,5 |
| tElaboracion mínimo (min) | 10 | 4 | 6 |
| tElaboracion máximo (min) | 13 | 6 | 8 |
| tElaboracion moda (min) | 12 | 5 | 7 |
| tConsumo mínimo solo(min) | 10 | 7 | 11 |
| tConsumo máximo solo(min) | 16 | 9 | 16 |
| tConsumo moda solo(min) | 13 | 8 | 13 |
| tConsumo mínimo ac.(min) | 11 | 7,7 | 12,1 |
| tConsumo máximo ac.(min) | 20,8 | 15,6 | 20,8 |
| tConsumo moda ac.(min) | 15,6 | 9,6 | 15,6 |

**Tabla 3.3**: Propiedades de los menús.

| ingrediente | | pizza | ensalada | hamburguesa |
|---|---|---|---|---|
| 1 | pan (unidades) | 0 | 0 | 1 |
| 2 | carne (kg) | 0 | 0 | 0,1 |
| 3 | mozzarela (kg) | 0,05 | 0,01 | 0 |
| 4 | lonchas_queso (kg) | 0 | 0 | 0,01 |
| 5 | lechuga (kg) | 0 | 0,05 | 0,01 |
| 6 | harina (kg) | 0,1 | 0 | 0 |
| 7 | tomate (kg) | 0,02 | 0,05 | 0,01 |
| 8 | atún (kg) | 0,01 | 0,02 | 0 |
| 9 | aceite_de_oliva (L) | 0 | 0,01 | 0 |
| 10 | vinagre (kg) | 0 | 0,01 | 0 |
| 11 | aceitunas (kg) | 0,01 | 0,01 | 0 |
| 12 | jamon (kg) | 0,01 | 0 | 0 |
| 13 | mostaza (L) | 0 | 0 | 0,0050 |

**Tabla 3.4**: Cantidades de cada ingrediente necesarias para la elaboración de los menús.

| ingrediente | caducidad días | tamaño pedido (unid.) | precio €/unid. | Espera en los pedidos(min) | | |
|---|---|---|---|---|---|---|
| | | | | mínimo | moda | máximo |
| pan (unidades) | 1 | 504 | 0,2 | 30 | 40 | 50 |
| carne (kg) | 7 | 378 | 8 | 30 | 40 | 50 |
| mozzarela (kg) | 3 | 53 | 3 | 30 | 40 | 50 |
| lonchas_queso (kg) | 15 | 81 | 6 | 30 | 40 | 50 |
| lechuga (kg) | 1 | 15 | 2 | 30 | 40 | 50 |
| harina (kg) | 30 | 953 | 1 | 30 | 40 | 50 |
| tomate (kg) | 3 | 66 | 2 | 30 | 40 | 50 |
| atún (kg) | 1800 | 100 | 7 | 30 | 40 | 50 |
| aceite_de_oliva (L) | 300 | 100 | 2 | 30 | 40 | 50 |
| vinagre (kg) | 300 | 100 | 0,5 | 30 | 40 | 50 |
| aceitunas (kg) | 300 | 100 | 1 | 30 | 40 | 50 |
| jamon (kg) | 300 | 100 | 15 | 30 | 40 | 50 |
| mostaza (L) | 100 | 50 | 2 | 30 | 40 | 50 |

**Tabla 3.5**: Diversas propiedades de los ingredientes.

| | precio fijo pedido € | umbral sobre el tamaño del pedido(% ) | porcentaje sobre el tiempo de caducidad (%) |
|---|---|---|---|
| Para cada ingrediente | 5 | 10 | 90 |

**Tabla 3.6**: Propiedades comunes de los ingredientes.

| Tiempos de atención de los cajeros | | |
|---|---|---|
| t. atención mínimo (min) | t.atención máximo (min) | t.atención moda (min) |
| $0{,}1 \times tipo_{cliente} + 0{,}2$ | $0{,}2 \times tipo_{cliente} + 0{,}3$ | $0{,}25 \times tipo_{cliente} + 0{,}35$ |

**Tabla 3.7**: Ecuaciones para el retraso triangular de la atención de los cajeros.

| Parámetros de tiempo en desistir de la cola de entrada (minutos) | | |
|---|---|---|
| t. mínimo | t. moda | t. máximo |
| 15 | 25 | 20 |

**Tabla 3.8**: Parámetros del retraso triangular para desistir en la cola de entrada.

| Parámetros de tiempo en tomar datos del menú de 2ª o 3ª opción (minutos) | | |
|---|---|---|
| t. mínimo | t. moda | t. máximo |
| 1 | 2 | 3 |

**Tabla 3.9**: Retraso triangular para replantearse un segundo o tercer menú.

## 3.12.-Conclusiones

Dado el número de elementos que se van a incluir en el modelo, el elevado número de parámetros de configuración (como se pueden constatar por las tablas previas 3.1 a 3.9), las complejas relaciones entre ellos etc., cuando pasemos a implementar el modelo en Arena se sobrepasarán con creces las prestaciones de la versión académica. De esta forma cuando implementemos las dos versiones del establecimiento de comida rápida estaremos en

condiciones de comparar las prestaciones de la versión Simkit frente a la de Arena para un modelo de cierta complejidad.

# Capítulo 4.-Modelado orientado al proceso usando Arena

## 4.1.-Introducción

Se realiza la descripción de la implementación de los requisitos enunciados para el modelo de un establecimiento de comida rápida en Arena. El número de elementos deberá sobrepasar los permitidos en la versión académica, cosa que ocurre de forma amplia. Se hace una descomposición de la implementación en elementos relacionados lógicamente y se realiza la explicación de las funciones de dichos elementos, pasando a describirse los componentes estándar de los que constan y la configuración de los mismos. De forma general los parámetros de configuración de los bloques se han realizado dando valores a bloque de variables de Arena de forma que se pueden controlar de forma centralizada. Este proceder permite al usuario variar dichos parámetros en un sólo lugar del modelo en lugar de tener que recorrer bloque a bloque para realizar la modificación.

## 4.2.-Entidades

Para el modelado usando Arena se ha creado un sólo tipo de entidad que representa al grupo de clientes: “Entity 1”, se crea con retraso temporal de tipo exponencial parametrizado mediante el “Shedule” “ plan clientes”. “Plan clientes” toma valores de la columna 6 de la variable “plan_personal”. En plan personal están los ritmos máximos de generación de clientes a lo largo de las horas del día, para modular el efecto precio y calidad del servicio, se descartan

aleatoriamente los que están en exceso sobre dicha modulación (los resultados se aproximan a los que se generarían directamente con los parámetros ajustados). Las entidades “Entity 1” que representan a los clientes guardan ciertas propiedades como atributos:

1. “t_entrada”: valor del tiempo de simulación en que el cliente llega al establecimiento, se utiliza como referencia para calcular los tiempos hasta superar las distintas colas y procesos.

2. “tipocliente”: indica el número de personas de que constaba el grupo de clientes generado.

3. “come_mesa2”: con el valor 1 para este atributo se indica que cliente come en mesas fijas, en caso contrario (valor 0) comería en mesas configurables.

   En la fase central del proceso en Arena se generan tantas copias de la entidad como clientes individuales representa, y tras este proceso a cada una se les da valor a los siguientes atributos:

4. “menú”: toma el valor del menú que desea en el primer intento el cliente.

5. “menú1”: toma el valor del menú que desea el cliente caso de no haber ingredientes para su primera elección, e intenta una segunda opción.

6. “replanteamiento”: toma el valor del nº de intentos que se han hecho para generar el menú del cliente, 0 si hay ingredientes para el primer intento, 1 si se probo con un segundo menú y 2 si se prueba una tercera vez, no hay mas posibilidades.

7. “ingrediente”: cuando se pasa a recabar los ingredientes de un menú, se realizan pasadas chequeando desde el ingrediente 1 hasta el 13, este atributo va tomando dichos valores.

8. “intentando_elaborar”: atributo auxiliar para la fase de selección de ingredientes, inicialmente se le da el valor 1, si falta algún ingrediente este atributo se pone a 0, indicando con esto que se pasa al modo chequeo de los ingredientes restantes para el menú pedido, es decir se mira si hay el stock conveniente, si no hay se lanza un pedido, y a diferencia de cuando tiene el valor 1, no se retiran las cantidades necesarias para la elaboración del menú. Al final del proceso se mira su valor, si es 1 se han podido retirar las cantidades de ingredientes necesarias y el menú se elabora, si es 0, falta de alguno y no se elabora el menú.

9. “pasada”: atributo auxiliar para hacer las copias necesarias de la entidad en función de los clientes que representa, así como para fusionarlas cuando se agrupan los pedidos.

10. “dur_comida”: se utiliza para codificar en un entero los menús tomados por los hasta diez clientes que forma un grupo, de forma que cuando se fusionen para formar el pedido total conservemos la información de los menús elaborados para cada cliente del grupo. La codificación se hace de la siguiente manera. Cada hamburguesa (menú tipo 3) generada añade 111, cada ensalada (menú tipo 2) añade 1 y cada pizza (menú tipo 1) añade 11, si no se elaboró menú por no haber ingredientes no se añade nada.

11. “t_comer_max”, “t_comer_med”, “t_comer_min”, guardan los parámetros necesarios para la espera triangular del proceso que representa el tiempo en comer, cuando se agrupan los menús en función de los menús realmente elaborados (analizando dur_comida) y de el número de personas del grupo, se dan valor a estos 3 atributos.

## 4.3.-Recursos

se ha considerado oportuno implementar el modelo utilizando los recursos siguientes, todos se planifican mediante módulos Shedule que permiten usar variables .

1. “cajero”: los disponibles se toman del “Schedule” “plan cajeros” que se nutre de la columna 5 de la variable “plan_personal”

2. “mesa” según el “Schedule” “plan_mesas” que toma su valor de la ecuación que sigue donde, “toman_mesa2” es una variable que tiene en su primera fila el número de mesas fijas (mesas2) planificadas en la simulación, siendo 30 el número total planificado.

$$plan\_mesas = 30 - toman\_mesa2[1] \tag{4.1}$$

3. “cocinero” utiliza el “Shedule” “plan cocineros” que se nutre columna 1 de la variable “plan_personal”

4. esp_pizza, esp_hamb y pinche, son recursos que permitirían modelar otro tipo de personal de cocina, pero en las simulaciones realizadas no se ha hecho uso de ello, se nutren de columnas de plan_personal que se han llenado con ceros.

5. “mesa2” utiliza el “Shedule” plan_mesas_fijas que se nutre del valor de la primera fila de “toman_mesa2” ; este recurso representa el número de mesas fijas que se planifican y que se ha tomado como variable de optimización.

Los parámetros para los bloques de construcción, Schedules, variables globales para cálculos intermedios y las que van a albergar los resultados de la simulación se han centralizado en el módulo variables, en general multidimensionales:

“clientes”: con 6 filas y 10 columnas en la primera fila alberga las probabilidades para cada tipo de cliente, de acuerdo con la Tabla 3.2, en las demás filas almacena resultados.

total clientes generados, fila 2; total clientes que entran al establecimiento, fila 3; clientes que desisten por haber muchos en cola(5), fila 4; clientes que dejan la cola por superar el tiempo de espera, fila 5; menús originados, por cada tipo de cliente fila 6.

“p_cola 4”: 1 columna y 4 filas, propiedades de la cola de entrada; en la primera fila el número de clientes en la espera que hace que se desista, y las otras tres filas los parámetros de espera triangular de los clientes, con los valores recogidos en la Tabla 3.8.

“plan_personal”: con 24+2 filas (una para cada hora del día) y 6 columnas; la columna 2 el plan horario de cocineros, la columna 5 el plan horario de cajeros y la columna 6 plan inicial de llegada de clientes. Las columnas 1, 3 y 4 no se usan en la simulación. La fila 25 tiene el valor medio, de las 24 filas anteriores y la 26 el coste por hora de cocinero y cajero.

“t_espera”: 1 columna y 3 filas, en ella se guardan los tiempos totales que invierten las entidades en llegar a los hitos siguientes; la fila 1 tiempo hasta obtener mesa, la fila 2 el tiempo hasta hacer pedido superando el cajero, la fila 3 tiempo hasta obtener el pedido y pasar al consumo del mismo y finalmente la fila 4 el tiempo hasta consumir el menú y abandonar el establecimiento.

“prop_menus”: 4 filas y 14 columnas, la fila 1 se ocupa del menu 1(pizza), la 2 del menu 2 (ensalada) la 3 para la hamburguesa y finalmente el 4 se usa en algunas columnas para recoger los datos si no se consumiese ningún menú. Las columnas recogen datos de entrada o bien resultados de la simulación; a continuación se relata el significado de cada columna; la columna 1, las probabilidades acumuladas de elección de cada menú; la 2 el precio; las 3, 4 y 5 los tiempos de elaboración para la espera triangular de acuerdo con los valores de Tabla 3.3; las 6, 7, 8, 9, 10 y 11 los parámetros de las esperas triangulares de consumo de los menús contenidos en la Tabla 3.3; en la columna 12 están los menús realmente elaborados, en la 13 los fallidos por falta de ingredientes y finalmente en la 14 los solicitados inicialmente.

“datos_econ”: 1 columna y 5 filas, en la fila 1 se van acumulando los gastos de personal en la simulación, en la 2 los costes de las compras de materias primas, en la 3 el valor de los menús vendidos, en la 4 el beneficio como resta de las dos primeras columnas a la 3ª; y finalmente en la 5 el beneficio mas el valor del stock presente a precio de adquisición.

“prop_matPrim”: 13 filas y 19 columnas, cada fila se dedica a un ingrediente; el significado de las columnas es el siguiente: las columnas 1, 2 y 3 contienen las necesidades para cada uno de los 3 menús con los valores de la Tabla 3.4; la columna 4 tiene los precios, la 5 la caducidad, la 6 los tamaños de los pedidos, la 7 el nivel de existencias que lanza el pedido, la 8 el porcentaje de tiempo consumido sobre la caducidad que lanza el pedido, la 9 el precio fijo por pedido, las 10, 11 y 12 los parámetros de las esperas triangulares para el servicio de los pedidos (valores de las Tablas 3.5 y 3.6), la 13 las existencias en el stock “en consumo”, la 14 las existencias en el stock “en reserva”, la 15, el tiempo de compra de las existencias “en consumo”, la 16, el tiempo de compra de las existencias “en reserva”, la 17 los pedidos en curso, la 18 los pedidos realizados y finalmente la 19 la materia prima desperdiciada por haber caducado.

“total_grupos_comen”: contiene el número de grupos que finalizan la comida en el establecimiento.

“t_med_esp”: con 4 filas contiene los valores medios de los tiempos de espera que, al igual que “t_espera” contiene los tiempos globales.

“plan_cli_inic”: 26 filas y 1 columna, contiene el plan de llegada de clientes inicial, se copia en el inicio con los valores de la columna 6 de plan_personal.

“n_hastaEmpezar”: contiene el número de grupos de clientes que han obtenido el menú y se disponen a comerlo.

“plan_sin_servicio”: 26 filas y 1 columna, contiene el valor del plan de clientes tras ponderar el efecto del precio sobre el plan inicial (todavía no se ha ponderado el efecto de calidad de servicío de ahí el nombre.

“menu_fallido”: variable auxiliar que contiene el menú que ha resultado fallido por faltar ingredientes

“prob2”: 3 filas, contiene el ajuste de probabilidad para los dos menús disponibles cuando no hay ingredientes para el solicitado, se utiliza Ec. (3.5).

“retardo_2_opcion”: 4 filas, tras fallar el primer menú se debe esperar para el segundo las 3 primeras contienen los parámetros de la espera triangular contenidos en la Tabla 3.9 y el 4º la probabilidad para intentar 2ª o 3ª opción, que en las simulaciones se toma como 0.

“prec_prob_ideales”: 3 filas y 2 columnas. En la columna 1 están los valores de los precios ideales de los menús y en la 2 las probabilidades ideales para esos precios, sus valores se recogen en la Tabla 3.3.

“denominador”: variable auxiliar en la que se guarda el denominador de la Ec. (3.4) cada vez que calcula.

“toman_mesa2”: 3 filas, en la primera se sitúa el número de mesas fijas que se utilizarán en la simulación; en la fila 2 se acumulan los grupos de clientes que usan mesas fijas y en la fila 3 se guarda el número de personas que usan las mesas fijas.

“aux_ent_ser_num1” y “aux_ent_ser_num2”: variables auxiliares que guardan el número de serie de la entidad para luego buscar su copia en la cola de las mesas móviles, con el fin de ser retiradas por los módulos “remove”.

A continuación, se describe detalladamente el sistema, diagrama de módulos y módulos de datos

## 4.4.-Descripción de los Componentes del Proyecto

Los modelos que permite crear Arena constan de una serie de bloques que se conectan unos a otros para generar el modelo [Urqu06b] [Kelt02], los bloques se auxilian mediante elementos no gráficos, como bloques de variables, de entidades, shedules etc. Algunos de los parámetros de los bloques se definen abriendo diálogos pulsando sobre su representación gráfica. Esta manera de modificar los parámetros de los elementos es incómoda si en distintas simulaciones se tienen que modificar varios elementos. Con el fin de centralizar los parámetros susceptibles de ser modificados, se definen en los bloques de construcción mediante variables, de forma que

cuando se quieran modificar los parámetros de la simulación basta con modificar las variables que recogen dichos parámetros en lugar de ir recorriendo uno a uno los bloques afectados

## 4.5.-Generación de las entidades "Cliente" (ver la Figura 4.1):

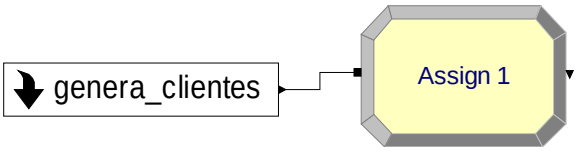


**Fig. 4.1**: Bloques de creación de clientes.

Los clientes se generan en la ventana principal en los dos módulos siguientes:

**El módulo "Assign 1"** asigna a la entidad cliente, los atributos:

tiempo de llegada al establecimiento, composición de personas del grupo,

e incrementa la variable global que recoge el número de grupos con esa composición que llegan al establecimiento.

## 4.6.-"genera_clientes"

es un submodelo con un salida por la que aparecen los clientes generados (ver la Figura 4.2) consta de:

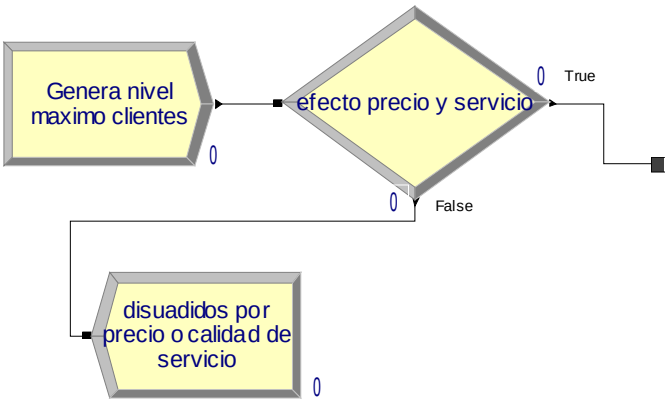


**Fig. 4.2**: Expansion de “genera_clientes”.

el modulo create “Genera nivel máximo de clientes ” los genera con distribución exponencial del tiempo de llegada al nivel máximo que se ha obtenido del estudio de mercado. Toma dichos valores de “plan_cli_inic”. En el módulo decide “efecto precio y servicio” se descartan al azar parte de los generados en función de la modulación aplicable por el precio medio y por la calidad del servicio. Para calcular dicha probabilidad de descartar se toma los valores de la columna 6 de “plan_personal” y se divide por “plan_cli_inic”; y en función de ella se desvían al módulo dispose “disuadidos por precio o calidad de servicio”, los que pasan salen del submodelo al azar y aproximadamente al ritmo que corresponde al estado del modelo.

## 4.7.-Modelo de la cola de entrada

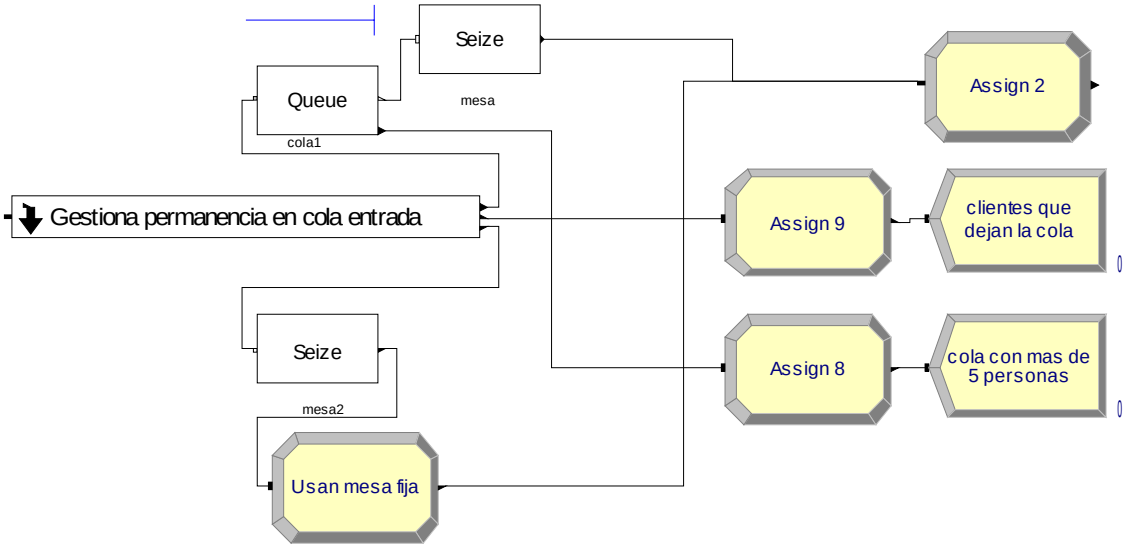


**Fig. 4.3**: Bloques que simulan la cola de entrada del establecimiento.

Es una de las partes más complejas del sistemas (ver la Figura 4.3). consta de un submodelo y de 9 bloques:

El funcionamiento global es el siguiente: los clientes llegan al submodelo “Gestiona permanencia en cola de entrada” (ver la Figura 4.4) , el cual los dirige bien a “Queue.cola1”, a “Seize.mesa2” o a “Asign 9”. Los que van a Queue.cola1 esperan en dicha cola hasta tener mesa configurable  o si hay más de 5 esperando no intentan entrar (balking) pasando a “Assign 8” donde se registra su número y son destruidos, si consiguen mesa configurable pasan a “seize.mesa” se ocupa las mesas y se pasan a actualizar las variables correspondiente en “Assign2”. Si se sale por “Seize.mesa2” se ocupa mesa de ese tipo y se registra el hecho en “Usan mesa fija” y se actualizan las variable s correspondientes. Los que van a “Assign 9” es por que desistieron por haber esperado demasiado en las colas, ahí se contabilizan y se destruyen en el dispose que sigue.

**“Queue.cola1**”, es el bloque gestiona la cola de entrada, tiene configurado un nivel de balking configurable, en las simulaciones se ha hecho de 5 grupos, los que desisten por haber más se redirigen al módulo “assign 8”. Los que atraviesan la cola de forma normal adquieren el recurso “mesa” en el módulo “seize.mesa”.

**“Seize.mesa”** bloque donde se toma el recurso “mesa”, se toma la cantidad necesaria en función del nº de personas presentes en el grupo: “AINT((tipocliente-.25)/2-1)+1”, tipocliente es el atributo indica dicha composición.

**“Seize.mesa2”**, bloque donde se toma el recurso “mesa2”, son las mesas fijas que no se pueden agrupar para albergar a grupos numerosos, por lo que sólo pueden ser ocupadas por los grupos de menos de 5 personas, cada grupo que entra ocupa sólo una mesa2.

**“Usan mesa fija”**, es un módulo assign auxiliar necesario para distinguir el tipo de mesa que ha ocupado el grupo, a su paso por dicho módulo a cada grupo cliente se le asigna el valor 1 al atributo come_mesa_2. También se actualiza variables globales para la contabilidad de los que usan las mesas fijas (el número de grupos y el número de clientes individuales).

**“Assign 8”**, incrementa la variable global que contabiliza número de clientes que dejan el establecimiento por haber más de 5 en la cola.

**“Assign 9”,** incrementa la variable global que contabiliza número de clientes que dejan el establecimiento por haber esperado más de lo admisible en la cola de entrada (ver Tabla 3.8).

**“Cola con más de 5 personas”**, modulo dispose donde finalizan su vida las entidades que representa a los clientes que abandonan por ser larga la cola, se ha fijado como parámetro 5 personas.

**“Clientes que dejan la cola”**, módulo dispose donde finalizan las entidades que desisten por esperar demasiado en la cola de entrada.

**“Assign 2”**, módulo donde se actualizan variables globales, como el número de clientes que consiguen acceder al establecimiento, el tiempo total que han invertido las que han superado la cola de entrada.

## 4.8.-“Gestiona permanencia en cola entrada”

Módulo complejo en el que se gestiona el tiempo de permanencia en la cola de entrada dando la posibilidad de desistir, así como la posibilidad de pasar a ocupar “mesas 2” a los grupos de menos de 5 personas en el caso de haber mesas de ese tipo libres. Todo ello por orden riguroso. Consta de 1 entrada y tres salidas y su composición es la siguiente (ver la Figura 4.4).

**“Decide 22”** separa a los grupos de más de 5 personas que solo pueden ocupar mesas móviles, y los dirige al modulo “separate 1”. los menos numerosos se someten a más procesos por lo que se dirigen al módulo “Decide 23”.

**“Separate 1”** hace una copia de la entidad cliente que le llega, una de las copias se dirige al módulo “Queue.cola1” a través de la primera salida, y la otra copia se pone en “Process 1” que es el primer elemento del control del tiempo de permanencia en la cola y la posibilidad de desistir si éste es muy largo.

**“Process 1”** modulo que genera una espera triangular a las entidades que le llegan, tras la espera se dirigen a “Decide 1”. La espera representa el tiempo admisible de espera en cola para los clientes. Los parámetros están recogidos en la Tabla 3.8

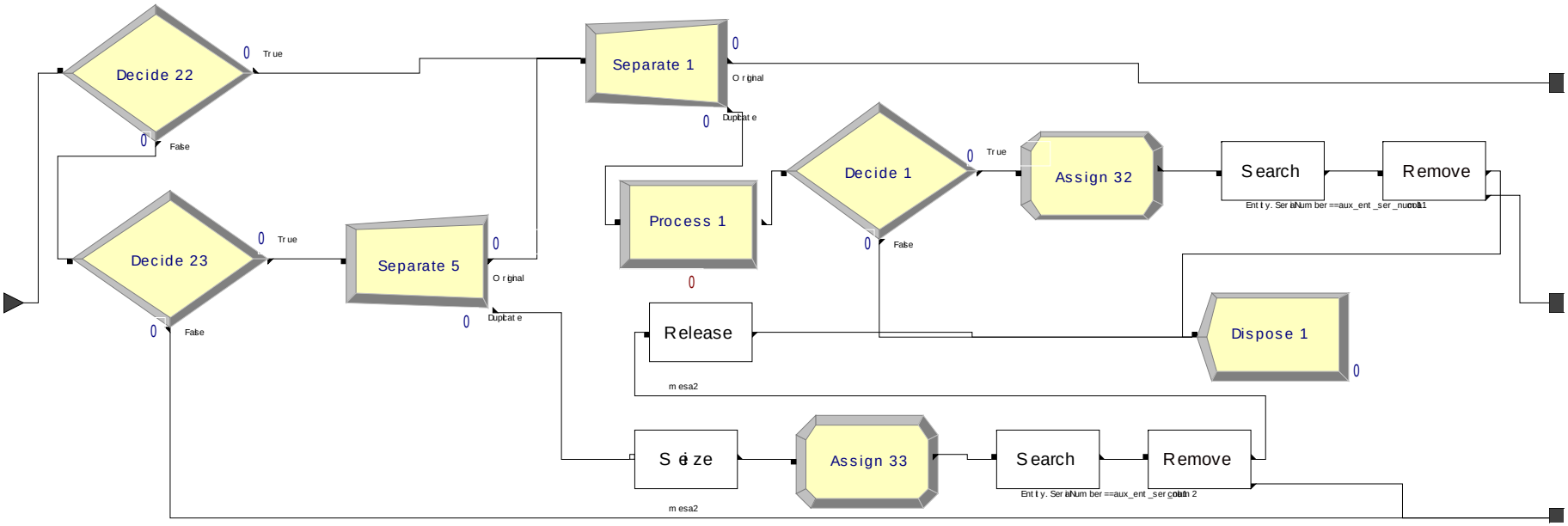


**Fig. 4.4**: Expansión del submodelo “gestiona permanencia en cola entrada”.

**“Decide 1”**, comprueba si hay entidades esperando en “Queue.cola1”, si no hay la gemela de la que se analiza ha entrado y por tanto no puede desistir, y como ésta ya no sirve para nada se destruye en Dispose 1. Si quedan entidades en la cola, se pasa a una comprobación más exhaustiva y se dirige al módulo Search situada en su salida “true”.

**“Asign 32”** guarda el “entity.serial.number”en la variable global “aux_ent_ser_num1”, luego se busca una copia de la entidad que tendrá el mismo numero de serie en cola 1 en el módulo “Search” que sigue.

**“Search”** realiza una búsqueda de la entidad gemela de la actual en “Queue.cola1”, si la encuentra se almacena su posición en la variable global “J” y pasa la entidad al módulo “Remove” siguiente.

**“Remove”**, recibe la entidad y el valor de la variable “J”, la entidad que ya no es útil se envía al módulo “Dispose 1” para su destrucción, se busca la entidad de la posición “J” de “Queue.cola1” que es la gemela de la que llega y como ha sobrepasado el tiempo de espera, se retira de “Queue.cola1” y se envía a la salida 2 de los que dejan la cola por esperar demasiado.

**“Decide 23”**, se comprueba si hay libres “mesa 2”, al tratarse de un grupo de menos de 5, si hay libre una mesa fija se pasa directamente a ocuparla, por lo que se envían a la salida 3. Si no hay “mesa 2”, pasan al proceso global de espera de “Queue.cola 1”, como los grupos numerosos y a uno especial para grupos de menos de 5, para lo cual la salida true dirige al módulo “Separate 5”.

**“Separate 5”**, divide el grupo de menos de 5 personas en dos entidades gemelas, una se envía al proceso que siguen los grupos numerosos en “Separate 1”, la otra al tratamiento especial de los grupos pequeños en el módulo “Seize.mesa2”.

**“Seize.mesa2”** en este módulo los grupos pequeños se encolan a la espera de recursos “mesa2” (a la vez una entidad gemela espera en la cola global “Queue.cola1”), si hay una libre se captura y se pasa al modulo “Asign 33” situado a su salida.

**“Asign 33”** guarda el “entity.serial.number”en la variable global “aux_ent_ser_num2”, luego se busca una copia de la entidad que tendrá el mismo numero de serie en cola 1 en el módulo “Search” que sigue.

**“Search”** busca en “Queue.cola1”, para encontrar si hay una entidad gemela de la que tenemos en curso, si la hay su posición se almacena en la variable “J” y se pasa al siguiente módulo “Remove”.

**“Remove”**, recibe la entidad y el valor de la variable “J”, la entidad que ya no es útil se envía al módulo release para liberar la “mesa 2” ocupada y luego a “Dispose 1” para su destrucción, se busca la entidad de la posición “J” de “Queue.cola1” que es una gemela de la que llega y si existe tal entidad se retira de la cola principal ya que hay una “mesa2” disponible para ella y puede saltarse la cola principal y ocupar dicha “mesa2” (que acaba de ser liberada por su gemela), para tal fin se dirige a la salida 3. Si no hubiese gemela en “Queue.cola1” “J” vale 0 y simplemente se libera “mesa 2”, (ya que la entidad gemela ha entrado en mesa configurable o bien ha desistido), y queda disponible para otras que esperan en la cola virtual de grupos de menos de 5.

**“Release”** modulo que libera la “mesa2” recién adquirida para dejarla a disposición bien de la entidad gemela encontrada en “Queue.cola1” bien de otro grupo de menos de 5 esperando en la cola virtual si la gemela hubiese  desistido por haber muchos en cola, abandonado por tiempo o entrado tomando una mesa móvil. Tras liberar el recurso, la entidad carece de utilidad y se envía a “Dispose 1”

**“Dispose 1”** aquí se destruyen todas las entidades gemelas auxiliares creadas para la gestión de los tiempos de espera y /o para gestionar la posibilidad de acceder con otro recurso (mesa2) por parte de los grupos de menos de 5.

## 4.9.-Modelo de la cola de cajero y selección de los menús individuales

En esta parte se generan los pedidos individuales de cada cliente de los que componen el grupo. Se espera en la cola de los cajeros, cuando están disponibles tras una espera se hacen los pedidos y a continuación se escinden en menús individuales, para ser tratados por separado en la parte de cocina donde se separa ingredientes y se elabora cada menú (ver la Figura 4.5).

Consta de tres módulos, dos bloques predefinidos y el tercero un submodelo.

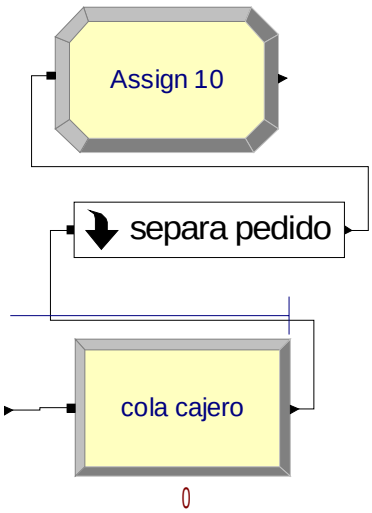


**Fig. 4.5**: Separación pedidos.

**“cola cajero”**, modulo de proceso, donde se captura el recurso cajero, se mantiene durante una espera triangular configurable y que depende del número de personas de que consta y se libera el cajero, su salida se dirige al submodelo “separa pedido”.

**“Assign 3”**, se ocupa en sumar al tiempo hasta hacer pedido el tiempo invertido por la entidad hasta llegar a el, y actualiza el atributo auxiliar “pasada” a 1, con este atributo podemos gestionar la división de la entidad original en tantas copias como individuos esta formada, para poder gestionar cada pedido de forma independiente. Tas asignar dicho atributo se dirige la entidad a “Decide 2”

## 4.10.-“separa pedido”

se ocupa de dividir la petición en tantos menús individuales como personas consta el grupo para que sean tratados por separado en los módulos que los generan. Su consta de una entrada y una salida y su estructura es ( ver la Figura 4.6).

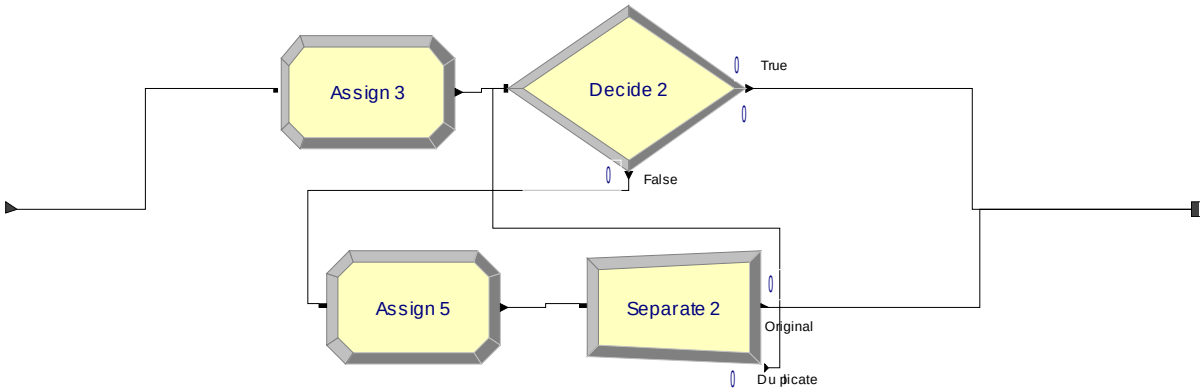


**Fig. 4.6**: Desglose de “separa pedido”.

.

**“Decide 2”**, compara el valor de los atributos “pasada” y “tipocliente”, si son iguales ya no hay que generar más copias y se dirige a la salida (esto ocurre si es tipo cliente=1 la primera pasada), si no lo es se manda a “Assign 5”, para generar las copias precisas.

**“Assign 5”**, incrementa pasada en una unidad y envía la entidad a “Separate 2”, para generar copias.

**“Separate 2”**, se crean dos copias iguales de la entidad que llega, una se envía a la salida y la otra se envía a “Decide 2” para que comparando los dos atributos “pasada” y “tipocliente”, se le haga o no pasar más veces para generar el número correcto de copias.

**“Assign 10”**, una vez transformados en pedidos individuales, se le asigna de forma aleatoria el tipo de producto que corresponde a cada uno, como las probabilidades son función de los precios relativos, se calculan previamente y son esos valores los que se aplican a cada menú.

## 4.11.-Gestión de los ingredientes y posibilidad de 2ª y 3ª opción

Se comienza por el ingrediente 1 y asumiendo que se va a elaborar el menú “intentando_elaborar”=1. Se mira si es necesario ese ingrediente para el menú, si no lo es se pasa al siguiente ingrediente (Figura 4.7). Si es necesario se intenta retirar la cantidad necesaria, si se consigue se procede con el siguiente ingrediente(si el stock esta por debajo del umbral o bien se sobrepasa el umbral de caducidad y no hay pedido en curso se lanza un pedido). Si no se mira si se esta haciendo pedido, en caso contrario se hace uno. Se le da a “intentando_elaborar” el valor 0 ya que el menú no va a ser elaborado; sin embargo se sigue con los demás ingredientes sólo comprobando si hay que hacer pedido de ellos. Cuando se llega al valor 13 de ingrediente se comprueba si “intentando_elaborar”=1(no han faltando ingredientes). Si es así se pasa a elaborar el menú con los ingredientes retirados, en caso contrario se pasa a decidir si se intenta otro de los menús o se deja sin elaborar.


**Fig. 4.7**: Bloques que gestionan 2ª y 3ª opción.

Consta de dos submodelos y un modulo predefinido el primero de los submodelos “toma ingredientes”, gestiona las variables que se representan a las materias primas, y los pedidos para reponerlas, y “Menu Segunda Opción” se ocupa de gestionar la posibilidad de que caso de faltar

materias primas para un tipo de menú, poder realizar un segundo menú diferente o incluso un tercero, o incluso no consumir producto alguno.

## 4.12.-“toma ingredientes”

Consta de una entrada y dos salidas, una de las salidas es para aquellos menús que es factible elaborar por haberse encontrado los ingredientes necesarios y se dirigen al siguiente bloque (“Decide 12”), si no se encuentran los ingredientes necesarios sale por la otra salida, dirigiéndose a “Menu Segunda Opcion”, para gestionar esta posibilidad. La composición de este módulo que consta de 8 bloques estándar y un submodelo es la siguiente (ver la Figura 4.8) .

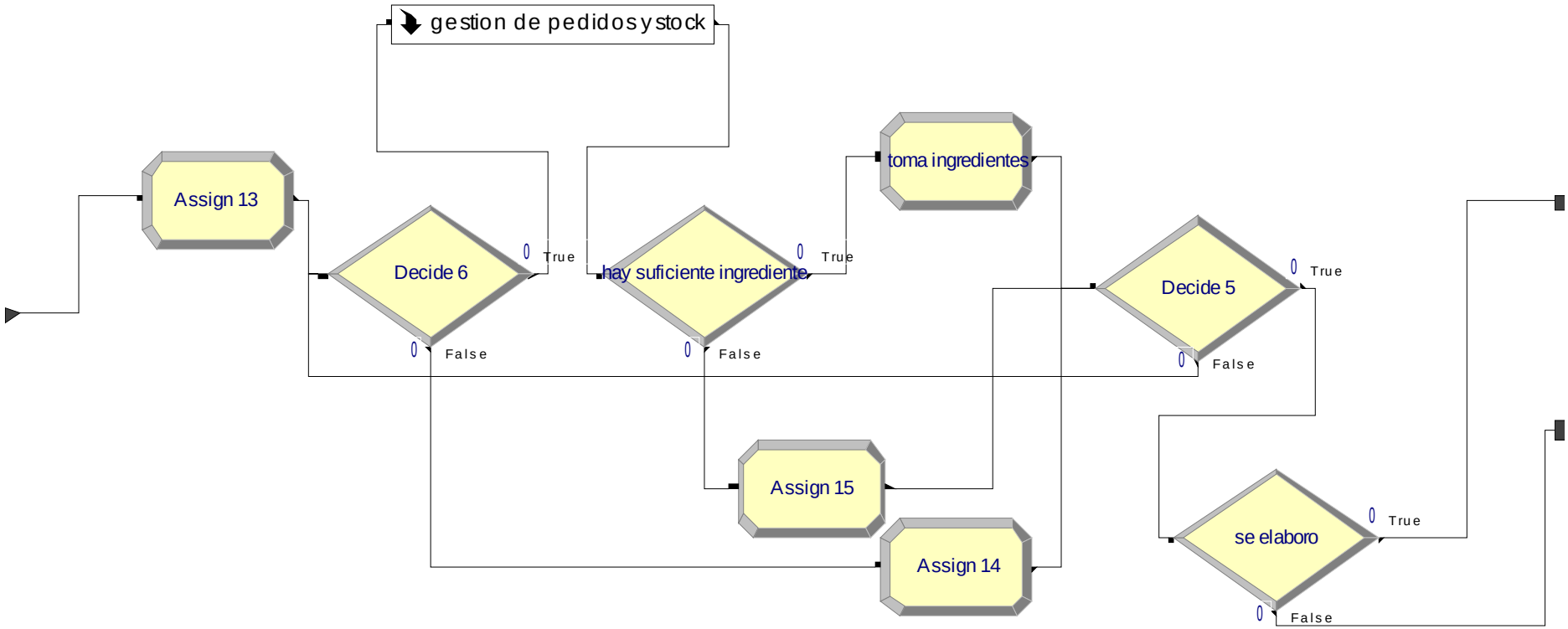


**Fig. 4.8**: Desglose en bloques del submodelo "toma ingredientes".

**“Assign 13”** se da valores inicial 1 a dos atributos auxiliares “intentando elaborar” e “ingrediente”, el primero sirve para que el resto de los bloques retiren la cantidad de producto para elaborar al menú o bien solo comprueben si hay que realizar pedidos; el segundo ( “ingrediente”) sirve para saber que producto es el que se está tratando, a su salida encontramos el módulo “Decide 6”.

**“Decide 6”** comprueba si se usa ese ingrediente para elaborar ese menú, si se usa pasa al bloque “GESTIÓN de productos y stock”, en el caso contrario , se dirige al bloque “Assign 14”.

**“hay suficiente ingrediente”**, comprueba que hay bastante stock para poder retirar la cantidad necesaria, si lo hay pasa al bloque “toma ingredientes”, si no hay pasa al bloque “Assign 15”.

**“toma Ingredientes”** se retira la cantidad necesaria y se pasa al siguiente ingrediente incrementando el atributo “ingrediente” y se va al módulo “Decide 5”.

**“Decide 5”**, se comprueba si se han tratado los 13 ingredientes, si ya se han tratado se va al módulo “se elaboro”, si faltan se repite el bucle yendo al módulo “Decide 6”.

**“se elaboro”**, comprobando el valor del atributo “intentando elaborar” se sabe si hay y se retiraron las cantidades necesarias, se va a la salida del producto elaborado y si faltó de algún ingrediente se va a la salida de los productos no elaborados.

**“Assign 15”**, si se llega a este módulo es porque falta de algún ingrediente, a partir de ese momento “intentando elaborar” pasa al valor 0, (se siguen comprobando los demás ingredientes pero no se retiran las cantidades necesarias), a su salida va al bloque “Decide 5” donde se itera hasta los 13.

**“Assign 14”**, se pasa al siguiente ingrediente, incrementando el atributo “ingrediente”, por este bloque se pasa caso de que el ingrediente no fuese necesario para elaborar el menú saltándose el camino más complejo, su salida es “Decide 5”.

## 4.13.-“gestion de pedidos y stock”

Es un submodelo con una entrada y una salida, a su paso las entidades originan la comprobación de existencias de materias primas, caducidades y si es necesario pedidos de las mismas, su composición es la de la (Figura 4.9).

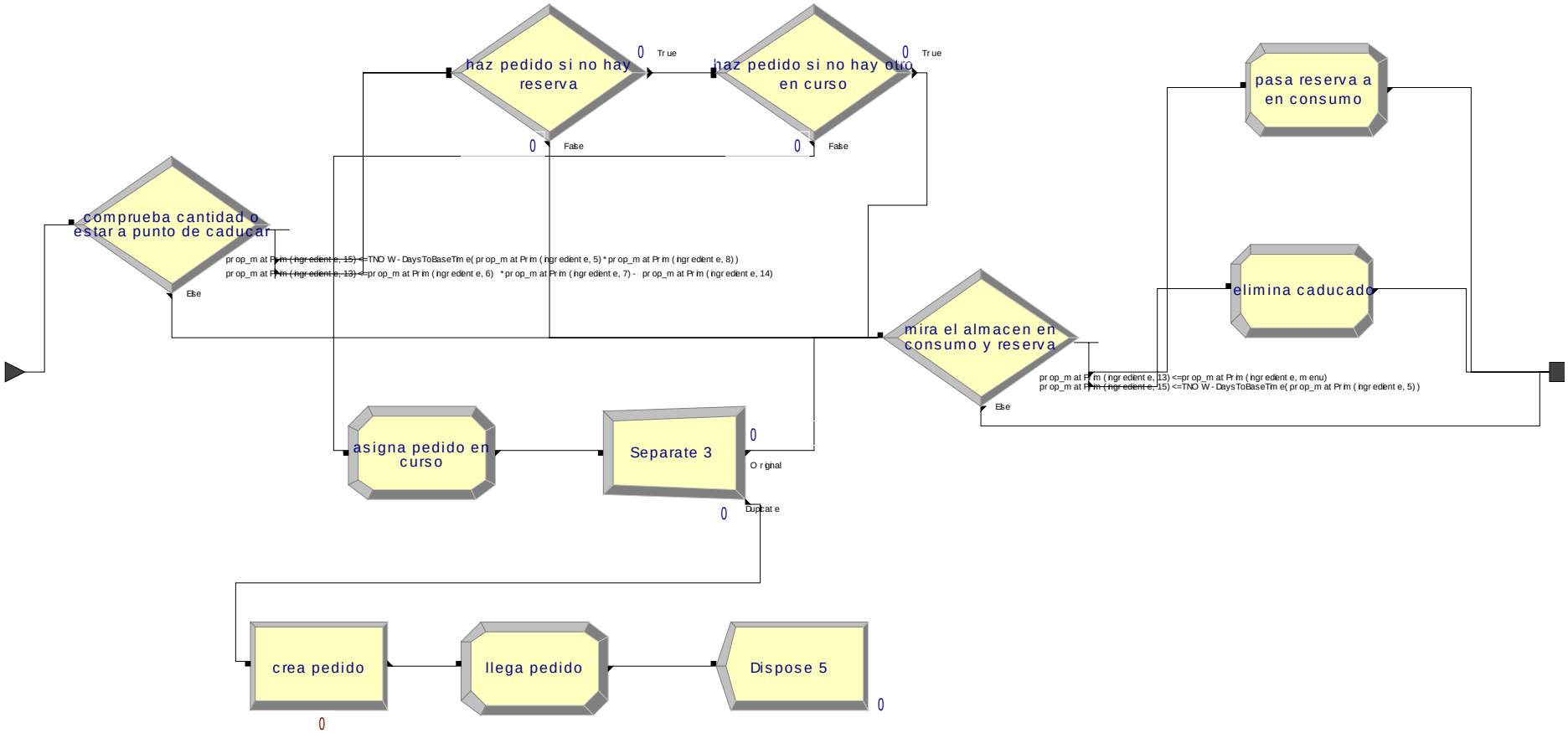


**Fig. 4.9**: Desglose en bloques del submodelo “gestion de pedidos y stock”.

**“comprueba cantidad o estar a punto de caducar”** este bloque decide, comprueba que las cantidades del stock están por encima del umbral para hacer pedido y si el tiempo hasta caducar es inferior del tope fijado (en el ejemplo la cantidad mayor que el 10% del pedio y el tiempo inferior al 90% del necesario para caducar). Si se sobrepasan esos límites se pasa a comprobar si es necesario hacer pedido. Si no se llega a esos límites se pasa a retirar las cantidades. Le siguen los módulos “haz pedido si no hay reserva” o bien “mira el almacén en consumo y reserva” respectivamente.

**“haz pedido si no hay reserva”** si los niveles en consumo son inferiores a los estipulados, puede ser que en reserva haya cantidades no consideradas en el módulo anterior, se comprueba si hay y caso de no haber se sigue con la idea de realizar pedido de la materia prima en el módulo “haz pedido si no hay otro en curso”. Caso de haber suficiente para no realizar pedido se va al módulo “mira el almacén en consumo y reserva” como en el párrafo anterior.

**“haz pedido si no hay otro en curso”** llegado a este módulo decide, se comprueba si hay pedidos en curso del ingrediente tratado, si lo hay no se realiza pedido y se va a “mira el almacén en consumo y reserva”, si no hay pedido se genera uno yendo al módulo assign “asigna pedido en curso”.

**“asigna pedido en curso”** se asigna valor uno al la variable que recoge si existe un pedido en curso y se pasa a gestionar éste para lo cual se va al módulo “Separate 3”.

**“Separate 3”** realiza una copia de la entidad, una de ellas sigue el curso del menú a realizar y se dirige al módulo “mira el almacén en consumo y reserva”. La copia representa al pedido recién realizado y va al módulo “crea pedido”.

**“crea pedido”** es un módulo process que se usa para introducir un retraso triangular que represente el tiempo de gestión del pedido. A su salida se supone que el pedido ha llegado y se dirige la entidad al modulo “llega pedido”.

**“llega pedido”** es un módulo assign , como cuando una entidad entra en el se supone que el pedido a llegado al establecimiento, se actualizan las variables que caracterizan los stock que corresponden al ingrediente manejado en la entidad, y se aumenta el número de pedidos realizados y se hace 0 la variable que indica si hay pedido en curso de dicho ingrediente. Tras salir de este módulo la copia de la entidad ya no es necesaria y se dirige al bloque siguiente “Dispose 5”.

**“Dispose 5”** la copia de la entidad ya ha rendido su utilidad para simular la gestión de un pedio y se elimina.

**“mira el almacén en consumo y reserva”** El stock global de una materia prima se gestionaba con dos grupos de variables, las del stock en consumo y la del stock en reserva. Aquí se pregunta si el stock en consumo va a ser suficiente para retirar la cantidad necesaria de ingrediente y generar el producto. Si no lo es se pasa al módulo “pasa reserva a en consumo”, la otra posibilidad es que el producto en consumo haya caducado y la salida lleva a “elimina caducado” y finalmente si no hay problema con la cantidad en consumo simplemente se sale del submodelo.

**“pasa reserva a en consumo”** si se llega a este módulo assign es porque la cantidad en consumo no es suficiente para elaborar el menú en curso, en el se pasa la cantidad “en reserva” a “en consumo”, se actualiza el tiempo de caducidad de en consumo y en reserva se pone a cero, una vez hecho esto se sale del submodelo.

**“elimina caducado”**, si se llega a este módulo es por que cierta cantidad de producto en consumo ha sobrepasado la fecha de caducidad, hay que eliminar la cantidad que reste y sustituirla por la cantidad en reserva actualizando la fecha de caducidad, se pone en reserva a cero y se actualiza la cantidad de producto desperdiciado. Tras todo esto se sale del submodelo.

## 4.14.-“Menu Segunda Opcion”

Es un submodelo en el que se gestiona la posibilidad siguiente. Si la salida de “toma ingredientes” es la que indica que al faltar de alguno no se puede realizar el menú en curso, se ofrece al consumidor la posibilidad de inclinarse por una segunda (o tercera) opción de menú, o bien no consumir nada. En este modulo primero se optará por consumir o no y caso de intentar una nueva opción se calculan las probabilidades de los menús no probados. Se introduce un retardo y se envía a la comprobación de ingredientes. Si se opta por no consumir nueva opción se toma la otra salida y se va al módulo “Assign 29”. Si se llega a la 4ª pasada siempre se sale sin consumir. Como la probabilidad entre intentar o desistir es configurable en la simulación se ha dado un valor de certeza a desistir. Este módulo tiene una entrada y dos salidas y su composición se detalla a continuación (ver la Figura 4.10) .

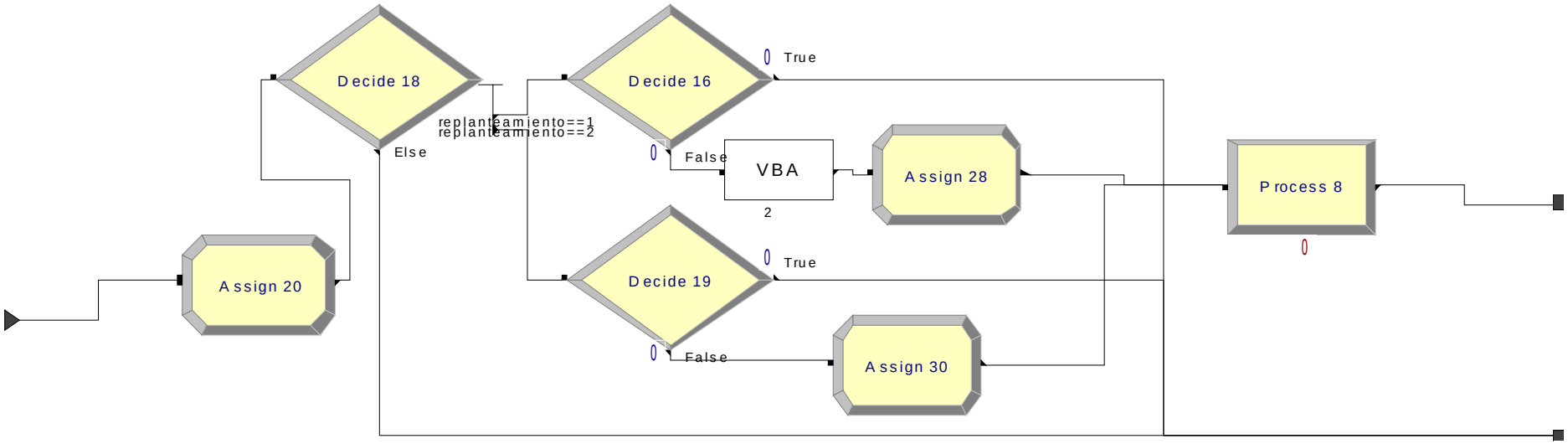


**Fig. 4.10**: Desglose del submodelo “Menu Segunda Opcion”.

**“Assign20”** incrementa el atributo replanteamiento de la entidad (así se sabe el número de veces que se ha intentado un nuevo menú), y se incrementa la variable global que lleva la contabilidad de los menús fallados (por tipo de menú), se pasa a continuación al bloque “Decide 18”.

**“Decide 18”** tiene 3 salidas, 2 de intentar nuevo menú, para replanteamiento 1 y 2; y la salida de desistir (siempre si es el tercer replanteamiento). La salida 1 lleva a “Decide 16”, la salida 2 “Decide 19” y la salida false lleva a la salida del subsistema en que no se genera el menú.

**“Decide 16”** aplica la probabilidad de desistir a los menús que se replantean por primera vez para decidir al azar , si se rechaza el replanteamiento, si se rechaza se sale del subsistema, si se decide replantear se va al siguiente bloque un bloque VBA2.

**bloque “VBA2”**, utiliza un programas Visual Basic para calcular las probabilidades aplicables para la elección de los dos menús restantes, se toman los valores vigentes de los 3 menús y se ajustan las de los dos elegibles para que siendo proporcionales a los valores que tienen siendo 3. los valores nuevos siguen siendo dos funciones de probabilidad, Ec. (3.5) luego se va al bloque “Assign 28”.

**“Assign 28”** se disminuye la variable que acumula los menús realizados del tipo inicial y se aumenta el del menú elegido mediante las probabilidades recalculadas, se asigna el atributo menú1, para tener en cuenta los dos menús no elegibles caso de una tercera reconsideración, a su salida vamos al bloque de retardo “Process 8”.

**“ Process 8”** se introduce el retardo definido para la elección de menús en segunda o tercera opción tras el retardo triangular se sale por la salida afirmativa.

**“Decide 19”** si es la segunda reconsideración se elige al azar el probar el tercer menú o bien desistir y salir por la salida negativa del subsistema. Si la elección es generar un nuevo menú, como en los atributos “menú” y “menú1” están los intentados anteriormente solo queda una opción que es la que se toma, se decrementan la variable global que contabiliza menu1 e incrementar la que contabiliza el menú elegido, tras esto se pasa al bloque de retardo ya comentado “Process 8”.

**“Assign 29”** en este módulo se contabilizan los menús que no se han generado al accederse a él por la salida negativa, tras pasar por el se accede al subsistema “junta pedido”.

## 4.15.-Módulo de generación de los menús

En función del atributo menú se pasa a los módulos proceso que introducen el retardo (ver Tabla 3.3) para la elaboración de cada menú y al final se guarda el menú elaborado y se actualizan las estadísticas ( ver la Figura 4.11).

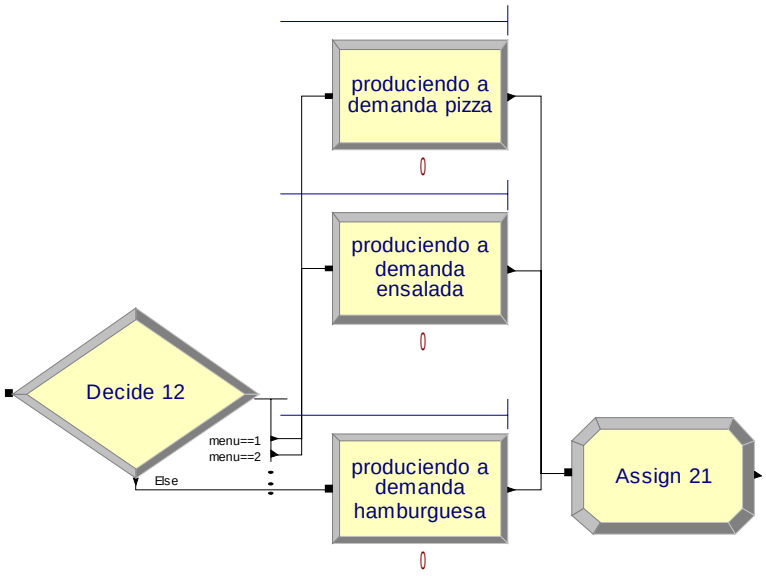


**Fig. 4.11**: Bloques que simulan la elaboración de los menús.

Esta parte simula los retardos en la generación de los distintos menús y consta de los bloques mostrados en la Figura 4.11.

**“Decide 12”** el menú a elaborar con los ingredientes tomados en el módulo anterior se dirige el módulo en el que se elabora, donde se encolará hasta tener recurso cocinero libre. Las salidas son los tres módulos donde se simula la generación de los tres menús a generar: “produciendo a demanda pizza”, “produciendo a demanda ensalada” y “produciendo a demanda hamburguesa”.

**“produciendo a demanda pizza”** modulo process con seize, delay, release; se captura un recurso cocinero, se mantiene ocupado el tiempo parametrizado para dicho menú (espera triangular) y se libera el recurso tras la elaboración del producto. Su salida va a “assign 21”.

**“produciendo a demanda ensalada”** modulo process con seize, delay, release; se captura un recurso cocinero, se mantiene ocupado el tiempo parametrizado para dicho menú (espera triangular) y se libera el recurso tras la elaboración del producto. Su salida va a “assign 21”.

**“produciendo a demanda hamburguesa”** modulo “process” con “seize”, “delay”, “release”; se captura un recurso cocinero, se mantiene ocupado el tiempo parametrizado para dicho menú (espera triangular) y se libera el recurso tras la elaboración del producto. Su salida va a “assign 21”.

**“Assign 21”** se actualizan la variable global que registra los menús elaborados, la que mantiene los ingresos obtenidos por ventas de menús y un atributo que permitirá obtener la composición en menús individuales de un pedido de varios menús “dur_comida” (en el modelo es de Interés sólo para fijar la duración de la comida ya que cada menú tiene una diferente). La

composición generando un número de 3 cifras en base (1+el número de personas del grupo mayor posible), en esta caso base 11, en el ejemplo que nos ocupa se un tipo de menú suma 111, otro suma 11 y el último suma 1. Si cuando se acumule el pedido este es mayor de 111 habrá al menos un menú del primer tipo, si su valor está comprendido entre 11 y 111 habrá al menos un menú de tipo 2 y ninguno de tipo 1; y finalmente si el valor es menor que 11 sólo hay menús de tipo 3; si valiera 0 no se habría generado ningún menú. Su salida va a "junta pedido".

## 4.16.-Subsistema que reúne los pedidos

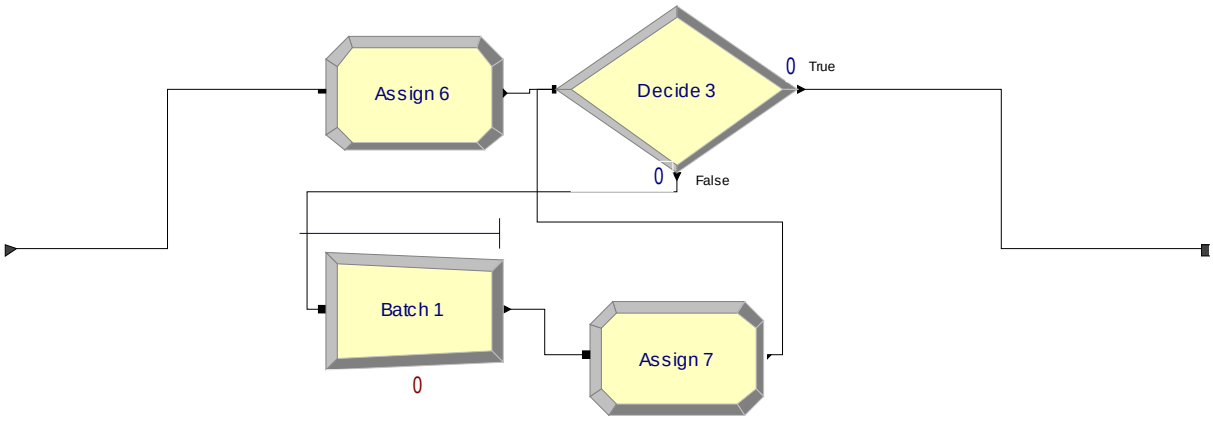


**Fig. 4.12**: Desglose del submodelo "junta pedido".

Tras generarse los menús individuales, se precisa reunirlos para suministrar al grupo de personas el pedido completo, también es necesario guardar información sobre la composición en menús individuales de pedido completo, esta información se codifica en un entero de hasta 4 cifras en la variable "dur_comida" en la que se suman los valores de los atributos de usuario de las entidades de un grupo de clientes (Fig 4.12). El subsistema tiene una entrada y una salida, con el siguientes componentes:

**"Assign 6"** se actualiza el atributo auxiliar pasada a l y a continuación dirige a la entidad al bloque "Decide 3".

**“Decide 3”** compara el atributo “pasada” con el atributo “tipo cliente” si son iguales se va a la salida y son diferentes dirige a la entidad al bloque “Batch 1”.

**“Batch 1”** en este bloque se reúnen las entidades por “Entity.SerialNumber” es decir las que son copias unas de otras, los atributos se suman al juntarse, y la salida se dirige al bloque “Assign 7”.

**“Assign 7”** divide por dos los atributos que van a seguir siendo útiles en fases siguientes: “t_entrada”, y “tipo_cliente”; ya que al sumarse el de la pareja en el módulo Batch 1 se habrían duplicado. “dur_comida” interesa que se sume ya que de esa manera se guarda en un número la codificación de los menús tomados por el grupo de clientes. A la salida se va al módulo “Decide 3”, donde el valor pasada se habrá ido incrementando hasta que se hayan reunido tantas entidades menú como personas tenía el grupo original, cuando se alcanza esto se sale del subsistema.

## 4.17.-Módulo generación de los tiempos de consumo del pedido

Su función fundamental es tomar la composición en menús del pedido del grupo y determinar el tiempo que durará la comida, asignando los atributos “t_comer_min”, “t_comer_med”, y

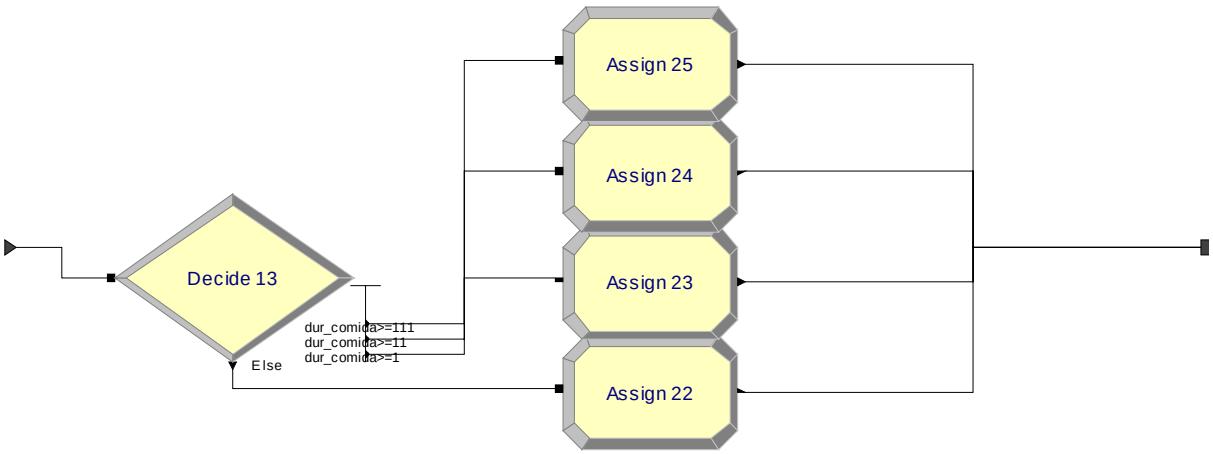


**Fig. 4.13**: Bloques que determinan el tiempo necesario para el consumo de los menús.

“t_comer_max” con los que se configurará una espera triangular (Figura 4.13).

Este bloque para conseguir el fin descrito consta del submodelo “asigna tiempo de consumo” se ocupa de asignar sus valores a tres atributos “t_comer_min”, “t_comer_med” y “t_comer_max” en función de la composición en menús de los pedidos y si el cliente está sólo o acompañado. Consta de una entrada y una salida y la composición en bloques es la siguiente.

**“Decide 13”** en función del atributo “dur_comida” que informa de la composición en menús del pedido se dirige al bloque “assign” adecuado que sigue. Su salida son los 4 módulos “assign”: “Assign 25”, “Assign 24”, “Assign 23” y “Assign 22”.

**“Assign 25”** da valor a los atributos “t_comer_min”, “t_comer_med” y “t_comer_max”, para el caso de haber elegido algún menú de duración en consumo máximo y teniendo en cuenta si el grupo esta formado por una o más de una persona.

**“Assign 24”** da valor a los atributos “t_comer_min”, “t_comer_med” y “t_comer_max”, para el caso de no haber elegido algún menú de duración en consumo máximo y sí alguno de duración intermedia y teniendo en cuenta si el grupo esta formado por una o más de una persona.

**“Assign 23”** da valor a los atributos “t_comer_min”, “t_comer_med” y “t_comer_max”, para el caso de haber elegido sólo menús de duración en consumo mínima y teniendo en cuenta si el grupo esta formado por una o más de una persona.

**“Assign 25”** da valor a los atributos “t_comer_min”, “t_comer_med” y “t_comer_max”, para el caso no haberse podido elaborar ninguno de los menús del pedido (en este caso los tiempos se hacen 0).

## 4.18.-Módulo final en que se consumen los productos

En esta fase se simula la fase del consumo de los menús, se determinan los tiempos medios hasta obtener el menú, parámetro de importancia para determinar la calidad del servicio que influye directamente en la afluencia de clientes al centro; también se calcula el tiempo total del servicio del completo. El aspecto del módulo es el de (Figura 4.14).

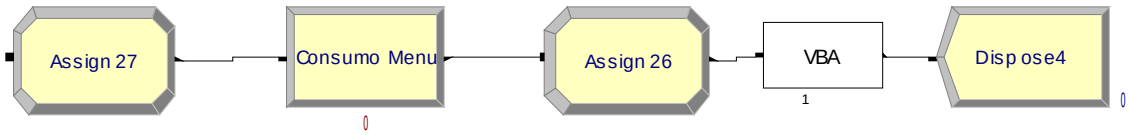


**Fig. 4.14**: Bloques que gestionan el consumo de los menús.

**"Assign 27"** en este bloque se actualiza el valor global del tiempo de espera hasta obtener el menú así como el de entidades llegadas hasta este punto de forma que módulos posteriores puedan disponer del valor del tiempo medio hasta obtener menú. Su salida se dirige al módulo "Consumo Menu".

**"Consumo Menu"** módulo "process" con funciones delay release, se introduce un retardo triangular parametrizado por los atributos de la entidad "t_comer_min", "t_comer_med" y "t_comer_max" que se calcularon en sistemas anteriores y se liberan la cantidad adecuada bien del recurso mesa o bien del recurso "mesa2", en función de los atributos "tipocliente" y "consume_mesa2". Su salida va al módulo "Assign 26".

**"Assign 26"** se actualizan variables globales como el tiempo total para el proceso, el cálculo hasta el momento de pasar por ahí la entidad del coste de la mano de obra, y del beneficio +

stock, para poder hacer un seguimiento en tiempo real si la simulación no es en modo “batch”, cálculo de los tiempos medios hasta los distintos hitos (pasar cola entrada, pasar cola cajero, obtener el pedido y tiempo completo). Su salida es el bloque “VBA1”.

**“VBA2”** modulo Visual Basic en el que se calculan los valores reales de llegada de clientes aplicando la influencia del tiempo hasta obtener el pedido a los datos de llegada a los que ya se ha aplicado el efecto de los precios. Su salida es el módulo “Dispose 4”.

**“Dispose 4”** completado el recorrido por el modelo, la entidad ya no es necesaria y se destruye en este módulo.

Juntando todos estos fragmentos y añadiendo elementos que permiten la visión de la evolución del modelo cuando las ejecuciones no son en modo “batch”, se obtiene el aspecto siguiente para la parte gráfica del modelo (Fig 4.15).

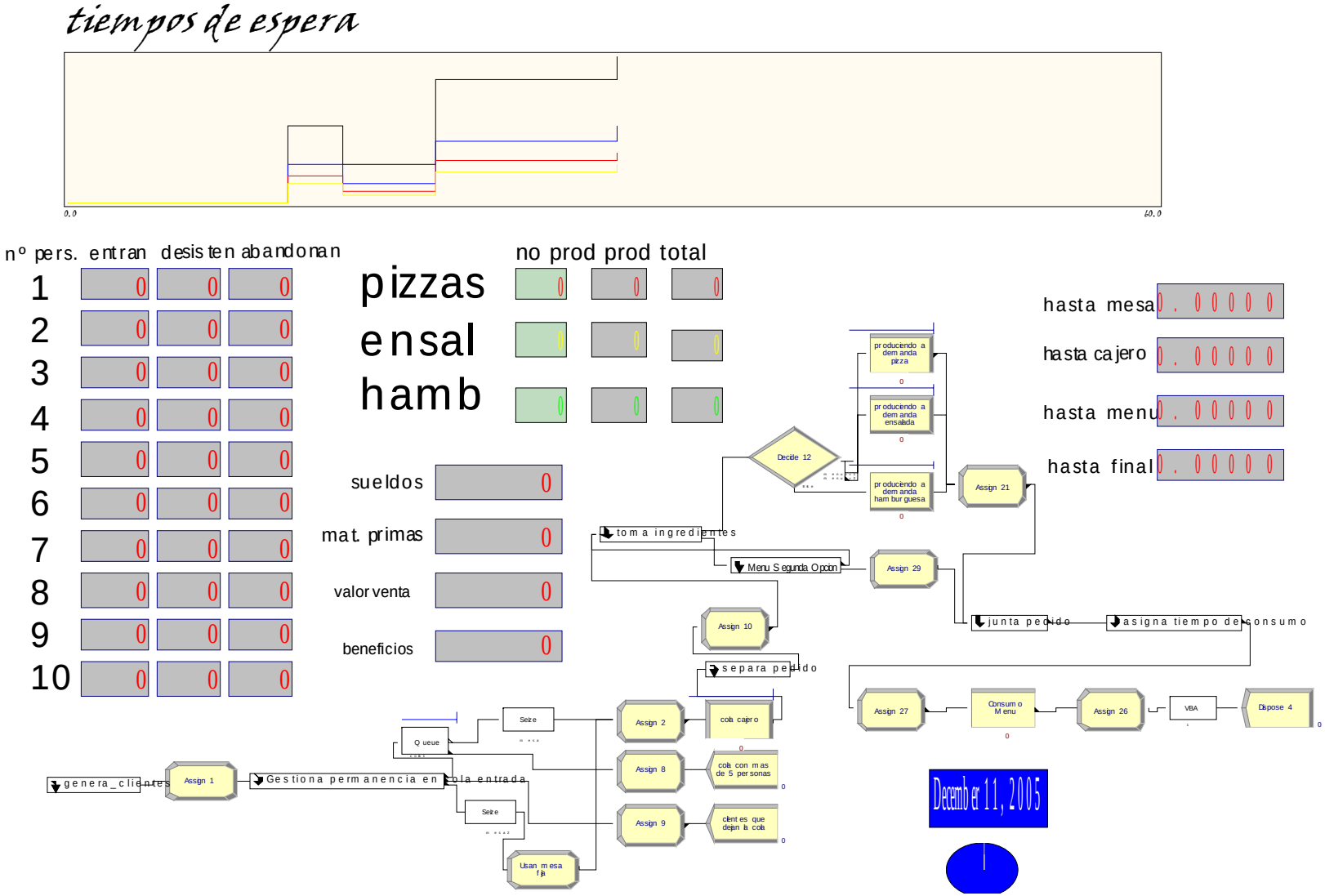


**Fig. 4.15**: Aspecto gráfico del modelo completo.

## 4.19.-Conclusiones

Se ha realizado el modelado en Arena, que consiste en la elección de bloques de construcción predefinidos y su ensamblaje para reproducir la lógica del modelo. Para realizar la representación con Arena se requirieron 82 bloques de construcción, algunos de los cuales se agruparon en submodelos. Se crearon 8 submodelos que reúnen bloques para formar agrupaciones que reflejaran lógicas parciales del modelo. Se crearon 6 colas y el modelo maneja 4 recursos diferentes. Dos de los recursos y las entidades que reflejan la llegada de clientes al establecimiento tienen una planificación variable a lo largo de las horas del día. Se han definido 20 variables, la mayoría de ellas multidimensionales, en las que se introducen los parámetros de las simulaciones o bien recogen resultados de las mismas. Todo este despliegue supera con creces las capacidades de la versión educativa de Arena.

# Capítulo 5.-Arena: diseño experimental, simulación y resultados

## 5.1.-Introducción

Una vez construido el modelo en Arena, se ha determinado el tiempo de simulación para que este alcance el estado estacionario. Se determina la función que produce el resultado óptimo, que consiste en maximizar el beneficio (ingresos-gastos totales) más el stock de materias primas a precio de coste. Se ensayan distintas configuraciones de los parámetros de entrada: tamaño del personal, combinaciones de los dos tipos de mesas y precio de venta de los productos. Se lanzan las simulaciones para combinaciones de dichos parámetros y se analizan los resultados. En el resto del Capítulo se describe en con mayor detalle este proceso.

## 5.2.-Diseño experimental

En los procesos de simulación se parte de parámetros de configuración que se toman como datos de entrada, a otros parámetros se les deja fluctuar y se buscan los valores que maximizan el objetivo, en nuestro modelo hemos considerado conveniente tomar los siguientes como parámetros de configuración fijos:

1. Afluencia máxima de grupos de clientes en función de la hora del día.

2. Composición de los grupos de clientes.

3. Comportamiento de los grupos de clientes en la cola de entrada.

4. Precios de los ingredientes y necesidades de los mismos para elaborar los productos.

5. Número de mesas en el local.

6. Tamaños de los pedidos, se parte de unos valores fijos.

7. Tiempos de elaboración de los productos.

8. Tiempo de atención de los clientes en la cola de cajeros.

9. Tiempos invertidos en el consumo de los productos.

10. La afluencia real de clientes depende de la máxima y de dos factores independientes menores o iguales a la unidad. El primero introduce la influencia del precio medio de los menús del establecimiento y el segundo la influencia de la "calidad de servicio" medida como el tiempo hasta obtener el menú deseado. Las funciones que permiten obtener dichos factores están definidas como parámetros de partida.

11. La probabilidad de elegir entre la oferta de menús, se fija unos valores a unos precios dados y se establecen funciones que penalizan en probabilidad la subida de precios de un producto y viceversa, también son parámetros de partida.

12. Coste por hora del personal.

Todo lo anterior se mantiene fijo entre simulaciones, pues se supone obtenido como datos de partida en estudios de mercado.

Se busca maximizar el beneficio. Como el establecimiento parte sin materias primas inicialmente, no es exactamente la diferencia entre ingresos por venta y gastos por adquisición de materias primas y gasto de personal lo que constituye el beneficio, sino que hay que tener en cuenta el valor del stock de materias primas que queda tras un periodo de simulación. Es por lo que se busca maximizar la suma entre el flujo de caja y el valor del stock de materias primas calculado con el precio de compra.

Parámetros que se varían para buscar la situación óptima:

1. El precio de cada uno de los menús.

2. La distribución entre mesas fijas y configurables, se asume que el número total de mesas que cabe en el local es de 30.

3. El personal que trabaja, concretamente el número de cocineros;

Aunque el modelo admite 5 tipos de personal diferente, para simplificar la simulación se maneja solo dos tipos de personal: cajeros y cocineros.

Dados los parámetros fijados, las necesidades de cajeros se reducen al mínimo es decir con sólo un cajero por hora se resuelven (solo se usa ~30% de la capacidad del recurso en las simulaciones) las necesidades, como menos de 1 no es posible, no queda otra opción que planificar 1 (en 3 de las horas punta hay dos ya que el horario sería de 8h y dos turnos a repartir en 13h abierto dan la posibilidad de 2 en esas horas, los fines de semana se cubriría con personal a tiempo parcial).

Los cocineros por las ligaduras que impone las 8h por día durante 5 días semanales, lleva a que si se quiere cubrir toda la semana con personal fijo, se contraten en bloques de 5 diarios,

con distribuciones de 8 h cada día con turnos rotatorios para abrir toda la semana. Probando con distintas combinaciones de los 5, que al ser parámetro entero en un número finito de pasos se llega al valor que da el máximo.

Se fijan dos tipos de mesas, las reubicables por los clientes y las situadas fijas y que no se pueden mover.

El número total de mesas es de 30 y se realizan simulaciones para las siguientes composiciones:

30 reubicables y 0 fijas;

25 reubicables y 5 fijas;

20 reubicables y 10 fijas,

Se fijan tres niveles de turnos para el personal, con una distribución horaria adaptada a los picos de la demanda:

8 grupos de 5 cocineros cada día;

7 grupos de 5 cocineros cada día;

6 grupos de 5 cocineros cada día;

Se varían los precios medios de los menús desde 7'32€ hasta 9'03€, para cada una de las combinaciones anteriores.

## 5.3.-Análisis de los resultados

Observaciones generales:

Como se comprueba de manera más amplia con la simulaciones realizadas en el modelo desarrollado en Java, el nivel alto de cocineros alcanza su mejor comportamiento en precios bajos, ya que los precios bajos mejoran la demanda y se consigue aprovechar mejor la mano de obra. Si se sube el precio cae la demanda y la mano de obra queda enseguida sobredimensionada. Por la misma razón los niveles bajos alcanzan su máximo rendimiento en precios mayores que los otros dos casos, ya que al haber menos mano de obra es preferible los precios altos que hacen caer la afluencia a los niveles en que se pueda atender con el personal (en caso contrario es el peor servicio es el que hace caer la demanda).

En las medidas realizadas, se comprueba que los beneficios máximos más altos se alcanzan en el nivel de 1 grupos de personal de cocina (Tabla 7.9).

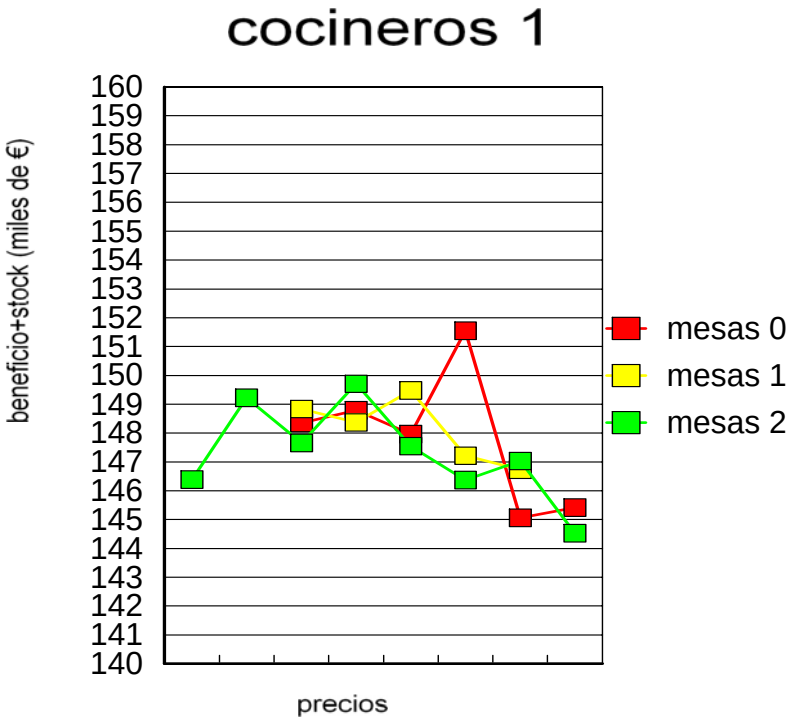


**Fig. 5.1**: Gráfica con los resultados de Arena en la zona de máximos.

Los niveles de mesas fijas tienen un efecto beneficioso tan sólo cuando la demanda es alta es decir a precios bajos. Según suben los precios empeora el comportamiento cuanto mayor es el número de mesas fijas que se usan. Tan sólo reservar 5 mesas fijas tiene un comportamiento similar a dejar todas reubicables en amplios rangos de precios y mano de obra. La razón es que al restar flexibilidad en niveles de demanda bajo, se perjudica a los grupos de clientes numerosos, cada uno que se pierde o se retrasa son muchos menús, por el contrario se beneficia a los grupos pequeños que aportan pocos menús.

Observando los resultados se ve que los valores más altos de los beneficios se alcanzan con 5 grupos de 7 cocineros diarios y utilizando todas las mesas configurables, concretamente en un precio medio de 8'23€, con una suma de beneficios y stock de 151549€ en 20 días de funcionamiento de la simulación, sin embargo en la zona optima no es muy sensible al número de mesas fijas planificadas siempre que estén entre 0 y 10.

Todos los demás parámetros de la simulación se mantienen fijos en las simulaciones.

En la Figura 5.1 se representa el beneficio+stock frente a los precios que conducen a valores máximos para el nivel de cocineros que conduce a máximos en Simkit, y con las combinaciones de mesas que llevan a valores máximos, sus valores provienen de la Tabla 7.10, en la Figura 7.10 se establece una comparativa de los dos modelos en la zona de máximos.

## 5.4.-Conclusiones

Una vez realizada la simulación se ha constatado que los valores óptimos son poco sensibles a las configuraciones de mesas utilizadas, siempre que se mantenga el número de mesas fijas por debajo de 10. Las entradas con niveles altos de personal alcanzan sus valores óptimos a precios

bajos ya que precisan afluencias grandes de clientes para compensar el mayor desembolso en gastos de personal. Las entradas con niveles bajos de personal alcanzan sus valores óptimos con precios altos. Esto es debido a que se prefiere disuadir el exceso de clientela por mayores precios en lugar de por peor calidad de servicio. La combinación de parámetros que produce el resultado óptimo en Arena es:

5 grupos de 7 cocineros diarios y utilizando todas las mesas configurables (las 30 mesas), concretamente en un precio medio de 8'23€, con una suma de beneficios y stock de 151549€ en 20 días de funcionamiento de la simulación.

# Capítulo 6.-Modelado mediante gráficos de eventos usando Simkit

## 6.1.-Introducción

Se pasa al diseño utilizando Simkit. El primer paso ha sido la construcción de la gráfica de eventos [Bank96]. Dada la complejidad del modelo (buscada a propósito para comparar las prestaciones de Arena frente a Simkit) no ha sido posible representar el diagrama completo. Se ha partido en dos partes, una gráfica (Figura 6.1) que incluye los eventos y las aristas, y otra que incluye el código que se escribe sobre las aristas y al pie del nodo en forma de texto indicando de forma clara la arista y el nodo a los que pertenecen. Del diseño se pasa de forma inmediata a la clase que extiende SimEntityBase que es la que realiza el trabajo de la simulación, el resto consiste en crear los objetos que representen a los distintos elementos del modelo (entidades, recursos colas etc.)así como los elementos necesarios para modificar de forma cómoda los parámetros de la simulación (crear una GUI) así como objetos que asistan en las tareas de guardar resultados [Fish78].

Una vez implementado el modelo usando Simkit se explica su funcionamiento del mismo, y de sus componentes gráficos.

## 6.2.-Diagrama gráfico de eventos

Analizando las características del modelo descritas en el apartado 3, se ha realizado el siguiente gráfico de eventos que lo representa (ver Fig 6.1), solo se han representado los nodos y las flechas sin los procesos, condiciones, tiempos y parámetros que completarían el diagrama ya que dado su tamaño no quedaría claro, si se representaran sobre el diagrama completo, por otra parte este da una idea clara de la dependencia de eventos y además algunas condiciones y procesos dada su complejidad no se prestan a su representación gráfica y es mas conveniente ponerlos en forma textual que a continuación listamos:

**doRun;**

doRun ➜ llegada,
(Incondicional)
retraso: t generación

doRun ➜ doGuardaIntermedios

/*transición para recoger información. En rojo en el diagrama (no es necesaria para el funcionamiento del modelo). Se planifican desde cero a varias, tantas como tiempos intermedios fijemos antes de lanzar la simulación*/

(Incondicional)
retraso: t planificados para muestreo de resultados

doRun ➜ doFinal

/*transición para recoger información. En rojo en el diagrama (no es necesaria para el funcionamiento del modelo)*/

(Incondicional)
retraso: t planificado para la simulación

**doCreacion:**

{si cola entrada con menos de 5 grupos pon entidad en cola entrada;
si cola entrada con menos de 5 grupos y si grupo es de menos de 5 personas aumenta la cola auxiliar de mesa2;
aumenta contadores clientes generados;}

doCreacion ➜ doCreacion

(Incondicional)
retraso:t generación

doCreacion ➜ doAbandonan

(Condicion: cola con 5 grupos;)
retraso:0

doCreacion ➜ doMesa;

(Condicion: cola con menos de 5 grupos;)
retraso: 0
Prioridad 1

doCreacion ➜ doMesa2;

(Condicion: cola con menos de 5 grupos y grupo con menos de 5 personas;
retraso: 0
Prioridad 2

doCreacion ➜ doDesisten

(Condicion: cola con menos de 5 grupos;)
Retraso: t desistir
parámetro: entidad.

**doAbandonan**

{incrementa contadores de clientes que abandonan}

**doMesa**

{si hay mesas configurables suficientes  Disminuye cola de mesa
y ocupa las mesas,
pon al grupo en la cola de cajeros;
aumenta contadores clientes usan mesa;
aumenta contadores clientes entran;}

doMesa ➜ doCajero

(Condición: hay suficientes mesas;)
retraso: 0;

doMesa ➜ doRetiraQM2

(Condición: hay suficientes mesas libre y el grupo es de menos de 5 personas;)
retraso: 0
parámetro: entidad que representa al grupo.
Prioridad 3

doMesa ➠ doDesisten

(Condición: hay suficientes mesas;)
retraso: 0
parámetro: entidad que representa al grupo.

**doMesa2**

{si hay mesas2 libres Disminuye las cola de mesa y cola de mesa2 y Ocupa mesa2
y pone el grupo en la cola de cajeros;
aumenta contadores clientes usan mesa2;
aumenta contadores clientes entran;}

doMesa2 ➜ doCajero

(incondicional)
retraso: 0

doMesa2 ➜ doRetiraQM

retraso: 0
parámetro: entidad que representa al grupo.
Prioridad: 4;

doMesa2 ➠ doDesisten

retraso: 0
parámetro: entidad que representa al grupo.

**doDesisten**

{aumenta contadores clientes desisten}

doDesisten ➜ doRetiraQM

parámetro: entidad que representa al grupo.

doDesisten ➜ doRetiraQM2

(Condición:  grupo de menos de 5 personas)

parámetro: entidad que representa al grupo.

**doRetiraQM**

{retira el elemento pasado como parámetro de la cola de mesas ya que entro por mesa2 o desistió}

**doRetiraQM 2**

{retira el elemento pasado como parámetro de la cola de mesas fijas ya que entró como mesa configurable o desistió}

**doCajero**

{si hay cajero libre retira el primer elemento de la cola del cajero y ocupa el cajero}

doCajero ➜ doSservicioCajero

(condición: si hay cajero libre)
retraso: t del servicio del cajero.
parámetro: entidad retirada de la cola.

**doServicioCajero**

{Atribuye los menús ;
Libera el cajero}

doServicioCajero ➜ doCajero

(incondicional)
retraso: 0;

doServicioCajero ➜ doTomaIngredientes

(incondicional)
retraso: 0;
parámetro: la entidad grupo de clientes.

**doTomaIngredientes**

{para cada menú elegido por una persona de las que componen el grupo:
mientras hay cantidad suficiente de cada uno de sus ingredientes se retira la cantidad necesaria hasta detectarse un ingrediente que falta, de los ingredientes necesarios posteriores no se toma cantidad alguna ya que no se elabora el menú}

doTomaIngredientes ➜ doLlegaPedido

(si al tomar los ingredientes se encuentra alguno en las condiciones para realizar el pedido;)
retraso: t para realizar pedido;
parámetro: el ingrediente que falta

doTomaIngredientes ➜ doTomaIngredientes

(si no hay ingredientes para hacer el menú solicitado y se decide probar otro diferente)
retraso: tiempo fijado para intentar 2ª o 3ª opción de menú.

doTomaIngredietes ➜ doCocineros

(si hay ingrediente suficiente se pone en el menú en cola de cocineros)
retraso: 0

doTomaIngredietes ➜ doComerMenu

(Si no hay ingredientes y se decide no probar otra opción de menú)
retraso: 0

**doLlegaPedido**

{Actualiza los datos de las existencias con la llegada del pedido: costes, caducidades etc...}

**doCocinero**

{si hay cocinero retira el primer menú de la cola de cocineros}

doCocineros ➜ ServicioCocineros

(si hay cocinero, se ocupa)

retraso: t en elaborar el menú

parámetro: grupo de clientes, menú

**doServicioCocinero**

{Libera el cocinero;

marca el menú elaborado del grupo de clientes}

doServicioCocineros ➜ doCocineros

(incondicional)

retraso: 0

ServicioCocineros ➜ doComerMenu

(si todos los del grupo están elaborados)

retraso: 0

parámetro: grupo

**doComerMenu**

{actualizamos datos globales para estadísticas y para calcular la calidad de servicio de grupos hasta_obtener_menu y el factor calidad de servicio con los tiempos medios hasta este punto}

doComerMenu ➜ doFinComida

(incondicional)

retraso: t en consumir en función del número de clientes y menús

parámetro: grupo de clientes

**doFinComida**

{actualizamos los datos globales para estadísticas de clientes que completan el consumo liberamos las mesas ocupadas por el grupo}

doFinComida ➜ doMesas

(si ocupaban mesa configurables)

retraso: 0

doFinComida ➜ doMesas2

(si ocupaban mesas fijas)

retraso:0

**doGuardaIntermedio**

{ejecuta la recopilación de los datos de simulación hasta dicho momento de acuerdo a la configuración que se haya planificado y la deja preparada para guardarla}

**doFinal**

{ejecuta la recopilación de los datos de simulación finales de acuerdo a la configuración que se haya planificado, y la deja preparada para guardarla}

Se han suprimido los elementos sobre las aristas y el procesamiento de los nodos por razones de claridad. En la sección precedente se incluyen en pseudocódigo dichos elementos.

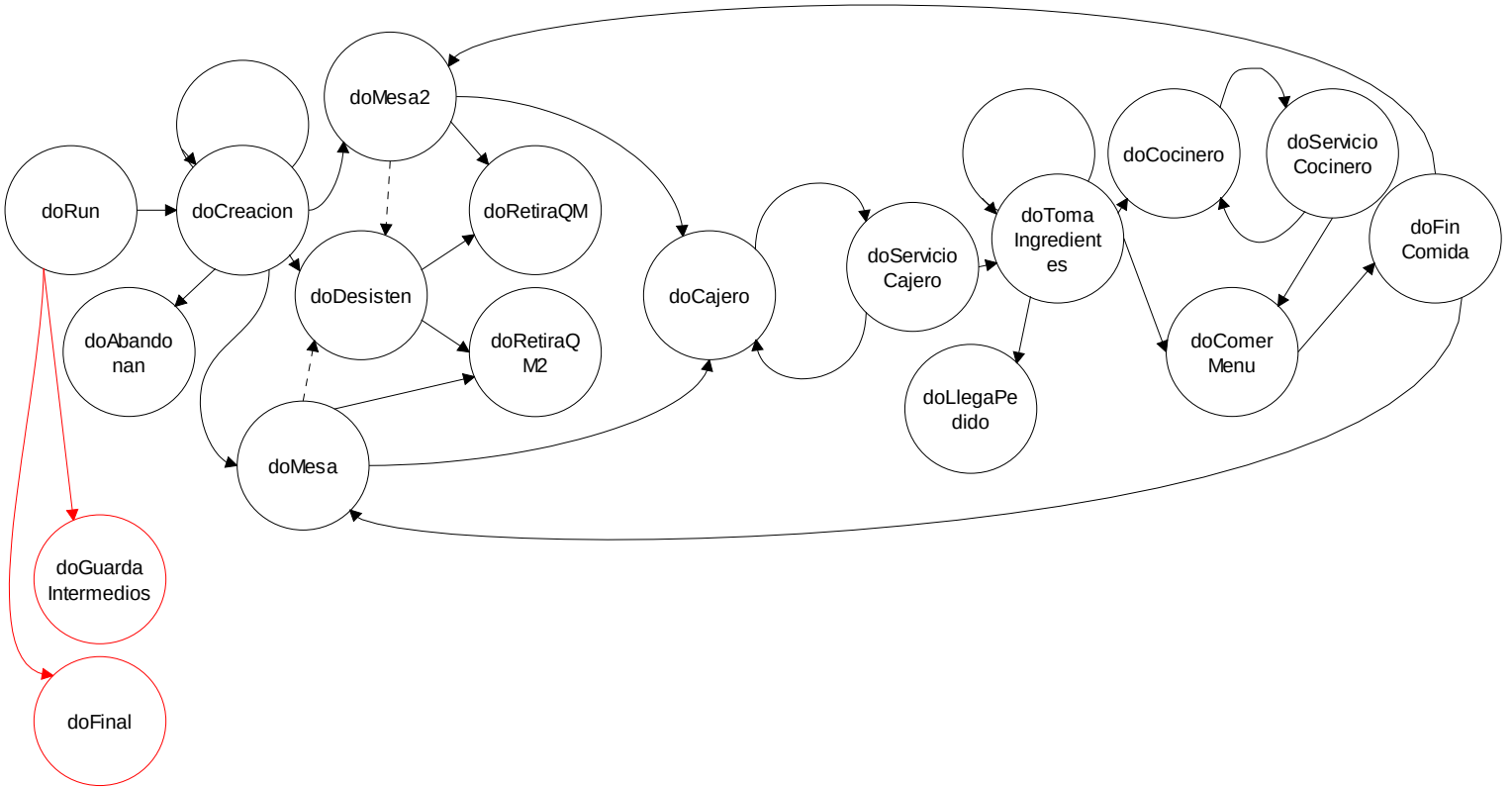


**Fig. 6.1**: Diagrama gráfico de eventos del establecimiento de comida rápida.

## 6.3.-Estructura de la aplicación

Un paquete auxiliar, para realizar funciones de carga y guardado de archivos así la manipulación de string: `accesorios`.

Una paquete auxiliar que contiene la GUI para la gestión cómoda de la aplicación y la visualización de sus resultados: `ventanas`.

El paquete con los elementos del modelo y la clase que realiza la simulación, así como clases que permiten integrar datos de configuración a partir de archivos de texto, y guardar resultados en archivos de texto: `examples`.

La aplicación se inicia, mediante la clase `BarraMenus` del paquete `ventanas`, que despliega una ventana con un conjunto de menús que permite configurar las opciones para la simulación, una vez realizada esta, se lanza la simulación (o un grupo de simulaciones) utilizando el método `iniciar()` de la clase `EstablecimientoComidaRapida`.

## 6.4.-Funcionamiento de la aplicación

La clase [Buss01b] que modela el establecimiento: `EstablecimientoComidaRapida` extiende `SimEntityBase` de la librería Simkit lo que le permite mediante los métodos `doNNN`, donde "NNN" son los nombres de los eventos, del diagrama de eventos (ver la Figura 6.1) modelándose de forma rápida el sistema. El proceso (entre {..} en el diagrama se modela como proceso dentro del método), los parámetros se reciben como parámetros objeto del método y las *edges* (flechas) se planifican en sentencias `if` , la condición es la condición del `if` y el evento a planificar se lanza usando el método `waitDelay` de la clase extensión de `SimEntityBase`. este método recibe como parámetros un String con el nombre (NNN) del evento a planificar, que luego se trata en

el método doNNN; un numero double, que contiene el tiempo que debe transcurrir hasta que se planifique; un array de Object que contiene los parámetros; y un entero que indica la prioridad para el caso de que se den eventos planificados para el mismo instante. Estos métodos están sobrecargados y admiten menos parámetros que los expuestos, son obligatorios el nombre y el tiempo de retraso. El diagrama de la Figura 2.3. se implementaría:

```
doA(..){
     h+=5; //procesamiento del nodo
     if (n>5)waitdelay("B", 3,new Object[]{j});//arista condicional, con tiempo=3, parámetro=j,
                                               //sin prioridad
     waitDelay("A",h);//arista incondicional con tiempo=h, sin parámetros ni prioridad
     if (h>3) interrupt("C", new Object[] { k });//arista condicional para revocar evento,
                                                 //parámetro=k
}
```

Las flechas discontinuas, que representan retiradas de eventos ya planificados de la lista de eventos futuros, se implementan con el método de la librería interrupt, que puede recibe como parámetros: el nombre del evento y un array de Object con los parámetros necesarios para decidir que evento del tipo pasado hay que revocar de la lista, (flecha A⇻C) .

En la clase EstablecimientoComidaRapida se han incluido métodos doNNN para tratar todos los tipos de eventos incluidos en el diagrama de la Figura 6.1, y con el cometido explicado en el texto del diagrama [Caho06].

Otros Elementos del modelo se han codificado como variables de instancia de la clase EstablecimientoComidaRapida como las colas de recursos, mesas fijas, configurables cocineros y cajeros; las variables que guardan datos económicos y temporales. Otros elementos con comportamiento más complejo como son los recursos: cajeros y cocineros, se han implementados como clases aparte que se crean una vez en EstablecimientoComidaRapida y se guardan sus referencias como variables de instancia. Cuentan con métodos para gestionar su

disponibilidad, el tiempo de servicio y sus costes. Son las clases: `PlanCajeros` y `PlanCocineros`. La gestión de materias primas se centraliza en la clase `GestionMateriasPrimas` que se ocupa de los stocks, caducidades, pedidos, retrasos en llegar pedidos, costes, materia desperdiciada .... La clase `ProbMenus` se ocupa de las propiedades de los menús: tiempos de elaboración , consumo precio, probabilidades de elección etc.

La simulación se inicia invocando el método `iniciar()` , que hace un `reset` del entorno invocando el método `reset()` de la clase `EstablecimientoComidaRapida`, en el cual se lanzan `reset` para todos los elementos del modelo que precisan volver a sus valores iniciales (en las clases creadas para representar cocineros, cajeros, etc se han añadido los métodos `reset()` correspondientes. A continuación invoca el `reset()` del entorno Simkit, fija el tiempo de simulación, ya que la condición de finalización es llegar a ese tiempo, e invoca el método de la librería Simkit que inicia la simulación `simkit.Schedule.startSimulation()`. Cuando se completa la simulación se llenan los datos de los paneles de la interfaz gráfica la Figura 6.2 con parte de los resultados.

La clase `GrupoClientes` que hereda de `GrupoClientesGenerico`, proporciona las entidades que representan a los clientes que llegan, para cada grupo nuevo se instancia un nuevo objeto `GrupoClientes`, dichas clases disponen métodos para obtener el tiempo de llegada, la composición en personas, y posteriormente los menús adquiridos, los elaborados los que faltan etc.

los clientes intentan ocupar los recursos `mesa` (o si son de menos de 5 personas `mesa2`), si existe el recurso se ocupa y se planifica `doCajeros`, se han elaborado dos colas para dichos recursos `mesaQ` y `mesa2Q` utilizando objetos `LinkedList` de la librería `java.util`, para medir el nivel de utilización de los recursos (los dos tipos de mesas, cocineros y cajeros) se ha seguido

el método general de usar una variable que guarda el tiempo total de uso de los ocupados y liberados y una lista en la que se anota el tiempo en que se ocupa un recurso, cuando se libera se calcula el tiempo transcurrido desde que entro en la lista hasta el instante en que sale; este tiempo se acumula al tiempo total de uso. Al llegar al final del tiempo de simulación se sacan todos los elementos presentes en la lista (corresponden a recurso ocupados pero no liberados) y se calcula el tiempo de ocupación de cada uno restando al tiempo final el tiempo en que entraron, estos tiempos se añaden también al tiempo total. En este momento se puede calcular el porcentaje de ocupación dividiendo el tiempo total obtenido por el número de recursos y por el tiempo total de simulación. También se va registrando en una variable el número máximo entes de un recurso ocupados simultáneamente.

Si hay 5 o más clientes esperando en la cola de entrada, se planifica el evento `doAbandonan`, donde finaliza su periplo tras actualizar las variable que recogen los clientes que abandonan.

los métodos `doDesisten`, `doMesa` y `doMesa2` son planificados para cada entidad de menos de 5 personas, y los dos primeros para las de más de 5, los dos últimos se planifican también cuando se libera una mesa del tipo correspondiente. El que se lleva a cabo en primer lugar para un grupo de clientes, (`doDesisten` lo recibe como parámetro y los otros dos lo obtienen de la cola de entrada correspondiente) provoca que el grupo de clientes sea retirado de las demás colas o se revoque el evento `doDesisten` planificado con ese grupo de clientes. Para retirar elementos de las colas de entrada se usan los eventos `doRetiraQM` y `doRetiraQM2`, que se planifican con tiempo de retardo 0 y sólo tienen dicho proceso, no planificándose desde ellos ningún evento futuro. Si se llevan a cabo `doMesa` o `doMesa2`, y se tiene éxito en ocupar el tipo de mesa correspondiente, se coloca el cliente en la cola de cajeros y se planifica un evento `doCajero` con retraso 0, también se actualizan los valores de las variables que guardan el tiempo total hasta

obtener mesa y el número de clientes que entran.

Del proceso de entrada, ya descrito se pasa a la cola de los cajeros, `doCajero` que retira un elemento de la cola de cajeros caso de estar disponible un cajero es ocupado y con el tiempo de servicio correspondiente se planifica `doServicioCajero`.

Cuando se lleva a cabo el evento `doServicioCajero`, se libera el cajero y se planifica de nuevo `doCajero` con retraso 0 se actualiza las variables que para el cálculo de las estadísticas de los cajeros, y se planifican tantos `doTomaIngredientes`, como personas forman el grupo, cada uno con el menú que corresponda. También se actualizan las variables que acumulan el tiempo hasta ser atendidos por el cajero y el número de clientes atendidos.

En `doTomaIngredientes`, se chequean las existencias de materias primas necesarias para el menú a elaborar, si existen suficientes se retiran y se pone en la cola de cocineros, con el menú a elaborar y se planifica un evento `doCocineros`, con retraso 0. Si no hay ingredientes suficientes se planifican los eventos `doLlegaPedido` para cada ingrediente con el retraso correspondiente. En función de la probabilidad de intentar otros menús se planifican bien `doTomaIngredientes` con otro menú y el retraso en la nueva petición o bien `doComerMenu` con el menú cero que indica que no se va a consumir.

En `doCocineros` caso de haber un cocinero libre se retira un cliente y menú de la cola y se planifica con el retraso correspondiente el evento `doServicioCocineros`. Cuando se lleva a cabo el evento `doServicioCocineros`, se libera el cocinero, se planifica con retardo 0, `doCocineros`, se actualizan las variables estadísticas de los cocineros y también se planifica con retardo 0 `doComerMenu` para el grupo de clientes correspondiente.

En `doComerMenu` se chequea si todos los menús del grupo se han elaborado (o si no se han

elaborado es por que se renunció a intentar otros menús), caso de estar el grupo completo se planifica el evento `doFinComida` con el retraso correspondiente, se actualizan las variables que guardan el tiempo hasta obtener todos los menús, los clientes que llegan hasta este momento y los valores promedio de tiempo que es un parámetro importante para modular el ritmo de llegada de clientes.

En `doFinComida`, se liberan las mesas ocupadas por el grupo, se actualizan las variables que acumulan el tiempo empleado hasta acabar de comer y el número de grupos que llegan hasta ese momentos; y las variables usada para el cálculo del tiempo de ocupación de las mesas.

En `doLlegaPedido` se aumentan las existencias de materias primas que corresponden al ingrediente que llega y se actualizan también lo precios y variables auxiliares que gestionan los pedidos realizados.

Hay dos tipos de eventos cuya utilidad es meramente estadística, su cometido es obtener y permitir guardar valores de parámetros del modelos en determinado instantes del tiempo de simulación, uno actualiza los valores en el tiempo final de la simulación: `doFinal`, y el otro hace la tarea para una lista de tiempos prefijada antes de iniciar la simulación: `doGuardaIntermedio`. Ambos se planifican en el evento inicial `doRun`, con los tiempos final y aquellos tiempos intermedios elegidos por medio de la interfaz gráfica.

## 6.5.-Interfaz gráfica de usuario

La aplicación se lanza desde una interfaz de usuario que tiene como clase principal `BarraMenus` que despliega un panel con el aspecto de la Figura 6.2. Como se puede ver consta de tres posibilidades de actuación: Archivo, Parámetros y Simulación.

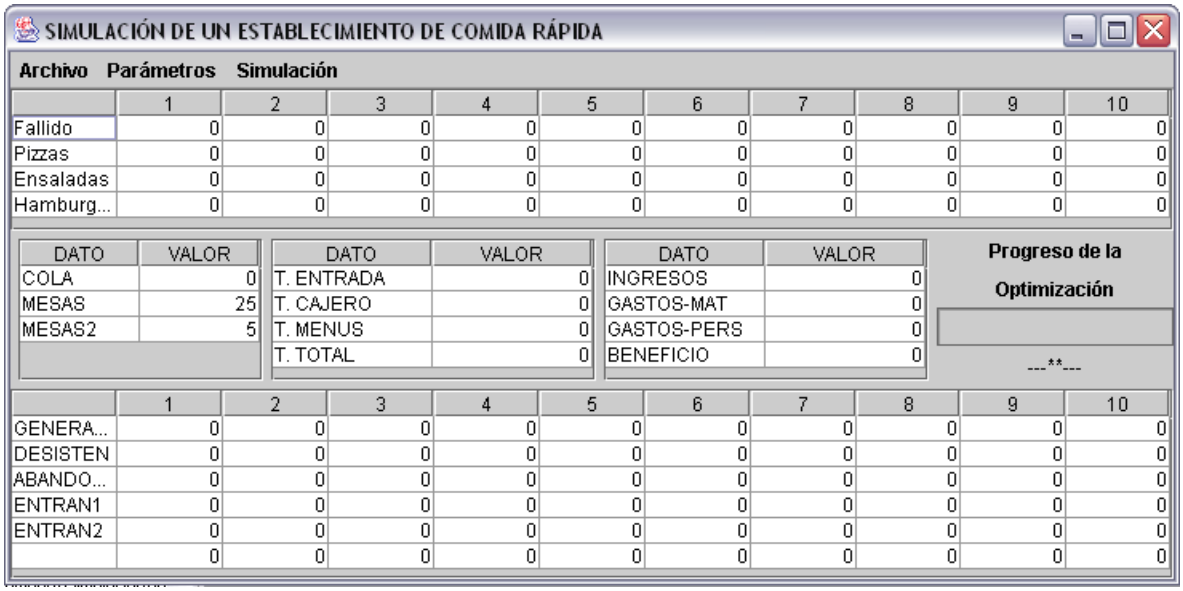


**Fig. 6.2**: Aspecto de la ventana principal.

En "Archivo" se pueden configurar las siguientes opciones ( ver la Figura 6.3):

En "Abrir" se abre un explorador de archivos que permite seleccionar el archivo que será la entrada de parámetros de configuración u optimización (según el modo en que se trabaje).


**Fig. 6.3**: Menú archivo.

En "Guardar" se abre un explorador de archivos que permite seleccionar el archivo donde se

guardara los resultados de la simulación u optimización.

En “Cerrar” se sale de la aplicación.

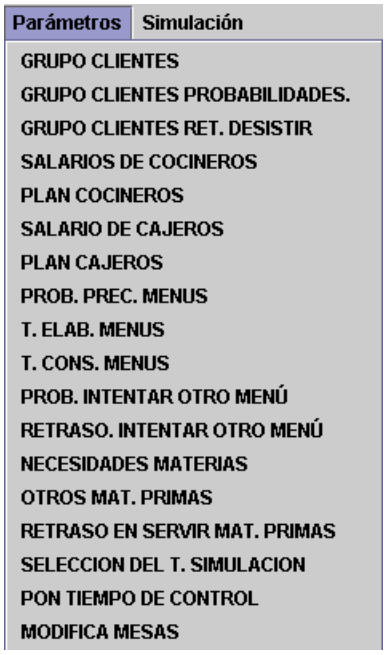


**Fig. 6.4**: Parámetros.

En “Parámetros” se pueden alterar los parámetros de configuración. En general se realiza mediante ventanas para introducir datos similares a la Figura 6.5, se pueden alterar los valores de las celdas y se marcan las casillas de verificación para que al pulsar aceptar se carguen los valores de las filas. Si no se marcan o se pulsa cancelar se mantienen los valores previos. Como ejemplo comentaremos las tres últimas opciones, las demás funcionan de forma similar, por lo que solo comentaremos tres opciones. El aspecto del menú desplegado se puede ver en la Figura 6.4.

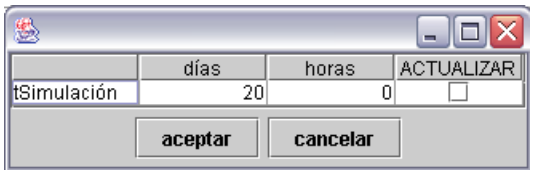


**Fig. 6.5**: Ventana que actualiza t. de simulación.

En “SELECCIÓN DEL T. SIMULACIÓN” se puede fijar el tiempo en días mas horas que durará la simulación, la ventana de dialogo es similar a la general y se muestra en la Figura 6.5.

En “PON TIEMPO DE CONTROL” se pueden añadir instantes de tiempo anteriores al planificado en el menú anterior para los que se guardaran los valores de los parámetros de simulación, la ventana que muestra el diálogo es ligeramente diferente de la general ( ver la


**Fig. 6.6**: Ventana “valores intermedios”.

Figura 6.6) ya que tiene un botón más “borrar lista” que permite transformar la lista de puntos intermedios en una lista vacía, es decir no se recogerían datos intermedios. El aspecto de esta ventana se muestra a continuación en la Figura 6.6).

En “MODIFICA MESAS” se puede cambiar el número de mesas fijas y configurables.

El resto de las opciones de Parámetros, presentan ventanas similares a la Figura 6.5 que permiten actualizar los parámetros correspondientes.

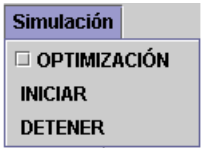


**Fig. 6.7**: Menú Simulación

El menú Simulación de la Figura 6.7 sólo tiene tres opciones. Una es del tipo casilla de

verificación “OPTIMIZACIÓN” que cuando está seleccionada pasa a modo optimización que consiste en lanzar simulaciones secuenciales con los parámetros de entrada que se han introducido a través de un archivo y se guardan los resultados de cada una de ella en archivos cuyo nombre se completa con indicativos numerales que indican el juego de parámetros utilizado. Como ejemplo si se indica para guardar el nombre “resultado”, los datos del juego de valores correspondientes del 2º lote de valores de cocineros( índice 1), del 3º lote de valores de mesas (índice 2) y del 17º lote de precios (índice 16) se guardarían como “resultado_1_2_16.txt”. Al concluir la optimización el resumen de resultados en el que estarían resaltados los valores óptimos se llamaría “resultadoRES.txt”. El modo normal se da cuando no esta seleccionada la casilla de verificación y sólo obtiene datos de una simulación, pudiéndose al final decidir si se guarda o no en un archivo cuyo nombre se decide en el diálogo de archivos ya descrito.

Las otras dos opciones permiten iniciar la simulación (normal u optimización) o bien detener el proceso iniciado.

Si el modo seleccionado es el de optimización, el avance del proceso se puede seguir en la barra de progreso, y al final del proceso se muestra en una ventana de texto el resumen de resultados que se guardan en el fichero seleccionado, con nombre “nnnRES.txt” donde nnn es el nombre seleccionado. Los resultados de cada simulación parcial necesaria para la optimización se guardan como “nnn_i_j_k.txt” donde i, j y k corresponden a los índices de la posición de los datos que se suministraron como datos para optimización en “nnn.txt”.

## 6.6.-Conclusiones

El proceso seguido para implementar este modelo de cierta complejidad utilizando las

librerías de Simkit no ha ocupado más tiempo que el dedicado a implementarlo utilizando la versión completa de Arena. De hecho la construcción de la clase corazón de la simulación `EstablecimientoDeComidaRapida.java` que extiende a `SimEntityBase`, es bastante inmediata a partir del diagrama de eventos. El resto de los elementos necesarios, bien para la simulación o bien para la interacción con el usuario, llevan más tiempo de desarrollo. Este se pude acortar si se crean objetos reutilizables que implementen las propiedades de entidades, recursos, colas etc. así como elementos para mostrar resultados o modificar parámetros en una interfaz de usuario (GUI). En esta dirección apuntan algunos objetos creados para el modelo simple comentado en el Capítulo 2 de la presente memoria.

# Capítulo 7.-Simkit: diseño experimental, simulación y resultados

## 7.1.-Introducción

Con la experiencia adquirida en la simulación utilizando el modelo de Arena, y dada la mayor flexibilidad del modelo de Simkit, ya que podemos programar directamente sobre el con Java, se diseña un proceso de optimización con un juego de parámetros más amplio que el utilizado con Arena. Se diseña la optimización de forma que esta sea totalmente automática. Los parámetros a ensayar se pasan en un archivo de texto y los resultados de cada ejecución son guardados de forma automática por el modelo de Simkit y cuando completa la combinación pasada guarda también en un archivos los datos y valores de las simulaciones que han conducido a resultados óptimos. Se han hecho un juego amplio de gráficas a partir de tablas con los resultados de las simulaciones realizadas en la búsqueda de la optimización.

## 7.2.-Diseño experimental

Se planifican como parámetros de configuración y variables de optimización los mismos que los tratados en el apartado 5 para la optimización con Arena. Los resultados observados siguen la misma tendencia y dan resultados absolutos comparables a los obtenidos con Arena. Para realizar de forma cómoda un juego de optimizaciones, se ha diseñado la aplicación en dos

modos, el modo normal en que tan solo se realiza una simulación y el modo "OPTIMIZACION" en el que se lee un archivo de texto con los parámetros variables de la optimización, la aplicación va lanzando una tras otra las combinaciones de configuraciones de optimización, va guardando los resultados con nombres de archivo que incluyen en el nombre indicativos que permiten saber que parámetros de optimización se utilizaron para su obtención, también genera un archivo resumen con los resultados globales, que incluyen el máximo global y máximos parciales para ciertos valores de los parámetros fijados en el archivo de optimización.

Dado el carácter automático del proceso se ha realizado una simulación más amplia que con Arena, en este último se ha explorado la zona más favorable que encontramos en la simulación amplia:

Se prueban 4 planificaciones de cocineros que corresponden a 8, 7, 6 y 5 grupos de 5 cocineros diarios, numerados de 0 a 3 (Tabla 7.9).

Se prueban 5 combinaciones de mesas fijas y configurables (0,30), (5,25), (10,20), (15,15) y (20,10); numeradas desde 0 a 4 (Tabla 7.8).

Se prueban 50 combinaciones de precios de los menús que van desde 4'3€ la pizza 3'4€ la ensalada y 5€ la hamburguesa en la combinación nº 0 hasta 9'2€, 8'3€ y 9'9€ en la combinación nº 49 con incrementos de 0'1€ en cada menú al pasar a la siguiente simulación(Tabla 7.5, Tabla 7.6 y Tabla 7.7).

Con lo cual cuando se lanza la optimización planificada se realizan 4*5*50=1000 simulaciones de forma automática. La combinación óptima obtenida está en la misma zona que con Arena.

## 7.3.-Análisis de los resultados

Los resultados obtenidos son (ver Tabla 7.1, Tabla 7.2, Tabla 7.3 y Tabla 7.4):

MÁXIMOS DE COCINEROS

0: MESAS: 1; PRECIOS: 31; VALOR: 147554.32500002242.

1: MESAS: 2; PRECIOS: 36; VALOR: 152627.22000002128.

2: MESAS: 1; PRECIOS: 36; VALOR: 148370.69500001927.

3: MESAS: 0; PRECIOS: 44; VALOR: 137685.09499996764.

MÁXIMOS DE MESAS

0: COCINEROS: 1; PRECIOS: 36; VALOR: 149604.21500002022.

1: COCINEROS: 1; PRECIOS: 38; VALOR: 150888.06000000113.

2: COCINEROS: 1; PRECIOS: 36; VALOR: 152627.22000002128.

3: COCINEROS: 2; PRECIOS: 42; VALOR: 141547.1900000371.

4: COCINEROS: 2; PRECIOS: 37; VALOR: 117938.46500001816.

MÁXIMOS GLOBAL

COCINEROS: 1; MESAS: 2; PRECIOS: 36; VALOR: 152627.22000002128€.

que corresponden a 7 grupos de 5 cocineros, con 10 mesas fijas y 20 configurables y a un precio medio de 8,13€ (similar al de 8,23€ de Arena, que también se da con 7 grupos de 5 cocineros). La relación mesas fijas/ configurables no es la misma que en Arena: 10 fijas con Simkit y 0 fijas con Arena, pero en la zona de máximos en ambas simulaciones no es muy sensible al número de mesas fijas siempre que estas estén entre 0 y 10, como se puede ver en las gráficas que representan los beneficios+stock frente a las variables de optimización (Figura 7.1, Figura 7.2, Figura 7.4, Figura 7.3, Figura 7.5, Figura 7.6, Figura 7.7, Figura 7.8 y Figura 7.9). En general en dichas figuras se observa que de 0 a 10 mesas fijas no tiene gran influencia en los resultados, 15 mesas fijas producen en general una caída ligera en beneficios+stock y usar 20 mesas fijas produce un gran descenso. En cuanto al nivel de cocineros, se observa que a menor número de cocineros los mayores beneficios se dan con precios más altos ya que es preferible que sean los precios los que disminuyan la demanda que no puede ser atendida por el bajo nivel de personal. de todas las maneras los niveles 0 y 1, vienen a coincidir en los valores máximos del objetivo para más o menos los mismos precios, el nivel 2 es claramente inferior y su zona de máximos esta por debajo de la de los anteriores y finalmente el nivel 3 es muy inferior y su zona de máximos esta muy baja con respecto a las otras tres.

Una comparación en la zona de máximos para Arena y Simkit se puede hacer en la tabla 7.10 y en la Figura 7.10.

| | cocineros 0 | | | | |
|---|---|---|---|---|---|
| | mesas 0 | mesas 1 | mesas 2 | mesas 3 | mesas 4 |
| 0 | 81474 | 81625 | 80234 | 77325 | 64923 |
| 1 | 84183 | 84808 | 83673 | 80205 | 68657 |
| 2 | 88011 | 87689 | 86669 | 83129 | 69536 |
| 3 | 91089 | 90833 | 90400 | 85471 | 75359 |
| 4 | 94167 | 93352 | 92385 | 88908 | 74926 |
| 5 | 97747 | 97051 | 96696 | 91992 | 77395 |
| 6 | 100438 | 100252 | 98913 | 94325 | 79036 |
| 7 | 102936 | 102585 | 102925 | 97224 | 80402 |
| 8 | 106986 | 105931 | 106081 | 100020 | 83992 |
| 9 | 105788 | 105390 | 103558 | 98080 | 82010 |
| 10 | 111497 | 111878 | 110266 | 105449 | 87440 |
| 11 | 113940 | 113463 | 113950 | 107508 | 87627 |
| 12 | 117498 | 116371 | 116717 | 108555 | 92088 |
| 13 | 119784 | 118165 | 118736 | 113830 | 93967 |
| 14 | 122678 | 120845 | 120691 | 114120 | 94130 |
| 15 | 123765 | 122916 | 122688 | 116112 | 96194 |
| 16 | 124945 | 125927 | 125447 | 117896 | 97889 |
| 17 | 129752 | 126783 | 126814 | 119713 | 97891 |
| 18 | 131831 | 130397 | 130225 | 120197 | 99274 |
| 19 | 131967 | 131705 | 131980 | 124354 | 103915 |
| 20 | 134951 | 133005 | 131157 | 123657 | 102793 |
| 21 | 136061 | 134078 | 132476 | 127085 | 102939 |
| 22 | 138429 | 136157 | 135108 | 129349 | 102461 |
| 23 | 138220 | 138338 | 137366 | 130065 | 105414 |
| 24 | 138933 | 139654 | 139478 | 129467 | 105651 |
| 25 | 143032 | 138864 | 139463 | 131026 | 107878 |
| 26 | 142234 | 143935 | 139985 | 132392 | 108879 |
| 27 | 141124 | 141457 | 141438 | 132295 | 111853 |
| 28 | 140708 | 146022 | 142612 | 133516 | 109947 |
| 29 | 146402 | 140973 | 139342 | 138774 | 107185 |
| 30 | 143864 | 144807 | 143835 | 134707 | 109923 |
| 31 | 143351 | 147554 | 143900 | 134215 | 111452 |
| 32 | 143577 | 146719 | 143892 | 138199 | 108495 |
| 33 | 143654 | 142730 | 145256 | 136453 | 110428 |
| 34 | 145096 | 146831 | 144139 | 133684 | 110220 |
| 35 | 143549 | 143497 | 143953 | 136137 | 110163 |
| 36 | 145572 | 145408 | 141335 | 138601 | 111268 |
| 37 | 143323 | 143106 | 144101 | 137646 | 103052 |
| 38 | 140296 | 140418 | 136075 | 137060 | 104913 |
| 39 | 144169 | 139199 | 141512 | 132478 | 110613 |
| 40 | 140167 | 142830 | 137563 | 135376 | 110142 |
| 41 | 138818 | 138932 | 142495 | 133260 | 106995 |
| 42 | 137740 | 140780 | 139665 | 131364 | 104182 |
| 43 | 133544 | 131153 | 132299 | 133590 | 99569 |
| 44 | 134680 | 134169 | 131252 | 130262 | 101462 |
| 45 | 135626 | 135256 | 134379 | 122282 | 102536 |
| 46 | 124713 | 131555 | 132186 | 129405 | 100914 |
| 47 | 127495 | 122065 | 131779 | 121248 | 101725 |
| 48 | 125225 | 123583 | 130052 | 116161 | 99729 |
| 49 | 119793 | 118421 | 121854 | 114265 | 102446 |
| max | 146402 | 147554 | 145256 | 138774 | 111853 |
| pos | 29 | 31 | 33 | 29 | 27 |

**Tabla 7.1**: Resultados del grupo de cocineros 0, ganancias en €.

| | cocineros 1 | | | | |
|---|---|---|---|---|---|
| | mesas 0 | mesas 1 | mesas 2 | mesas 3 | mesas 4 |
| 0 | 79150 | 79659 | 78827 | 75957 | 67740 |
| 1 | 82432 | 82908 | 82001 | 78907 | 67976 |
| 2 | 85917 | 85492 | 85659 | 81209 | 71113 |
| 3 | 88765 | 88998 | 87912 | 83824 | 74360 |
| 4 | 91829 | 92009 | 90453 | 87048 | 76735 |
| 5 | 94498 | 94972 | 94153 | 89803 | 79385 |
| 6 | 97208 | 97739 | 96992 | 93040 | 81760 |
| 7 | 100704 | 100500 | 100236 | 95059 | 83853 |
| 8 | 104081 | 104085 | 102736 | 98227 | 86672 |
| 9 | 103438 | 102769 | 102044 | 98629 | 84879 |
| 10 | 109614 | 108848 | 108501 | 103470 | 90102 |
| 11 | 111681 | 112254 | 111815 | 106481 | 91447 |
| 12 | 114880 | 115226 | 114504 | 110207 | 94610 |
| 13 | 118269 | 117028 | 116869 | 111798 | 96413 |
| 14 | 120317 | 120738 | 118824 | 114616 | 99089 |
| 15 | 123395 | 123525 | 122130 | 116746 | 98854 |
| 16 | 126549 | 125967 | 124859 | 119091 | 100537 |
| 17 | 128767 | 128365 | 128599 | 118344 | 101001 |
| 18 | 131033 | 130489 | 128362 | 122320 | 101968 |
| 19 | 133164 | 131536 | 132218 | 126163 | 104146 |
| 20 | 134939 | 136570 | 132164 | 126667 | 106430 |
| 21 | 137692 | 136848 | 134955 | 128807 | 104817 |
| 22 | 139090 | 137835 | 135993 | 130605 | 107825 |
| 23 | 139797 | 139511 | 137891 | 131908 | 107189 |
| 24 | 140993 | 139998 | 139734 | 131937 | 110229 |
| 25 | 143385 | 143882 | 143215 | 134783 | 111636 |
| 26 | 145323 | 144020 | 143724 | 134449 | 112994 |
| 27 | 147957 | 145582 | 143787 | 138238 | 113475 |
| 28 | 148628 | 145105 | 147660 | 138604 | 113036 |
| 29 | 147683 | 146865 | 147050 | 139272 | 111136 |
| 30 | 147084 | 147704 | 146058 | 140352 | 113008 |
| 31 | 146195 | 146983 | 149614 | 138742 | 114996 |
| 32 | 147958 | 147990 | 148743 | 138904 | 115845 |
| 33 | 148767 | 147855 | 144777 | 139387 | 116344 |
| 34 | 148627 | 147752 | 148604 | 140214 | 117332 |
| 35 | 147607 | 148655 | 147297 | 139289 | 115855 |
| 36 | 149604 | 144750 | 152627 | 138121 | 115509 |
| 37 | 144946 | 148349 | 145653 | 138544 | 116435 |
| 38 | 148122 | 150888 | 150262 | 138880 | 113987 |
| 39 | 141767 | 147812 | 144798 | 137968 | 111753 |
| 40 | 141730 | 147432 | 147223 | 137740 | 112695 |
| 41 | 141418 | 144431 | 146753 | 135581 | 106012 |
| 42 | 140407 | 143208 | 142217 | 138789 | 110423 |
| 43 | 137934 | 138212 | 139085 | 131782 | 112191 |
| 44 | 136022 | 139456 | 139609 | 132178 | 108259 |
| 45 | 135795 | 138973 | 137377 | 138463 | 106765 |
| 46 | 133951 | 136023 | 134956 | 123989 | 107553 |
| 47 | 129573 | 126197 | 126762 | 135760 | 104835 |
| 48 | 136326 | 119338 | 129734 | 130968 | 106511 |
| 49 | 133008 | 125730 | 128461 | 122921 | 107325 |
| max | 149604 | 150888 | 152627 | 140352 | 117332 |
| pos | 36 | 38 | 36 | 30 | 34 |

**Tabla 7.2**: Resultados del grupo de cocineros 1, ganancias en €.

| | cocineros 2 | | | | |
|---|---|---|---|---|---|
| precio | mesas 0 | mesas 1 | mesas 2 | mesas 3 | mesas 4 |
| 0 | 71118 | 70888 | 69614 | 67725 | 61644 |
| 1 | 73352 | 73897 | 73677 | 70932 | 63022 |
| 2 | 76978 | 76442 | 76261 | 72538 | 65535 |
| 3 | 79829 | 79102 | 78174 | 75885 | 67314 |
| 4 | 81716 | 82420 | 81873 | 78666 | 70340 |
| 5 | 84298 | 85073 | 83912 | 81313 | 72110 |
| 6 | 87670 | 87030 | 87260 | 82674 | 74255 |
| 7 | 90336 | 89817 | 89542 | 86415 | 75088 |
| 8 | 92671 | 92743 | 91950 | 87852 | 79036 |
| 9 | 92258 | 91553 | 91268 | 88693 | 79406 |
| 10 | 97307 | 97520 | 97091 | 92758 | 84010 |
| 11 | 100562 | 100380 | 99840 | 96518 | 84533 |
| 12 | 104164 | 104076 | 102481 | 98303 | 88661 |
| 13 | 106260 | 105821 | 105149 | 100715 | 91201 |
| 14 | 109047 | 107940 | 107441 | 102566 | 91554 |
| 15 | 111893 | 110893 | 111069 | 105771 | 94403 |
| 16 | 114405 | 113845 | 111947 | 108153 | 95613 |
| 17 | 116141 | 116153 | 114570 | 111076 | 96374 |
| 18 | 119378 | 118812 | 117371 | 111756 | 98710 |
| 19 | 121194 | 120169 | 118559 | 116011 | 101148 |
| 20 | 122594 | 123250 | 121472 | 116516 | 102240 |
| 21 | 125074 | 124222 | 123980 | 117650 | 102516 |
| 22 | 127093 | 127330 | 126925 | 121047 | 104149 |
| 23 | 129840 | 130334 | 128978 | 122698 | 104853 |
| 24 | 131362 | 132191 | 130329 | 125708 | 107057 |
| 25 | 133296 | 133710 | 131984 | 126354 | 106944 |
| 26 | 136689 | 134890 | 134001 | 127564 | 111223 |
| 27 | 135709 | 138229 | 136958 | 130217 | 113664 |
| 28 | 138014 | 139976 | 138302 | 130039 | 113419 |
| 29 | 139289 | 141069 | 138738 | 131897 | 112897 |
| 30 | 142488 | 142498 | 140637 | 133789 | 111791 |
| 31 | 142775 | 141800 | 142306 | 134696 | 116803 |
| 32 | 142844 | 143689 | 145407 | 134401 | 112630 |
| 33 | 144136 | 145093 | 144420 | 136524 | 115956 |
| 34 | 144897 | 145652 | 144439 | 138725 | 114930 |
| 35 | 146843 | 146112 | 145333 | 136885 | 116351 |
| 36 | 144246 | 148371 | 146719 | 139433 | 116984 |
| 37 | 145622 | 146972 | 147876 | 137693 | 117938 |
| 38 | 147026 | 145892 | 145397 | 138277 | 110729 |
| 39 | 143957 | 145730 | 144839 | 139720 | 112007 |
| 40 | 145956 | 146002 | 146922 | 136023 | 104710 |
| 41 | 141928 | 144790 | 147273 | 137783 | 114360 |
| 42 | 141665 | 146545 | 145964 | 141547 | 111452 |
| 43 | 145035 | 142349 | 139950 | 139133 | 110077 |
| 44 | 144560 | 140826 | 146182 | 135009 | 109944 |
| 45 | 143529 | 143272 | 142024 | 136916 | 111999 |
| 46 | 140807 | 143002 | 141517 | 133704 | 112610 |
| 47 | 138989 | 141145 | 132065 | 129173 | 112580 |
| 48 | 127530 | 141022 | 138717 | 129986 | 110364 |
| 49 | 132866 | 127134 | 139974 | 128812 | 112761 |
| max | 147026 | 148371 | 147876 | 141547 | 117938 |
| pos | 38 | 36 | 37 | 42 | 37 |

**Tabla 7.3**: Resultados del grupo de cocineros 2, ganancias en **€**.

| | cocineros 3 | | | | |
|---|---|---|---|---|---|
| | mesas 0 | mesas 1 | mesas 2 | mesas 3 | mesas 4 |
| 0 | 63292 | 62543 | 63487 | 60822 | 54318 |
| 1 | 65660 | 65482 | 65453 | 62721 | 55525 |
| 2 | 68662 | 67331 | 67808 | 64476 | 57849 |
| 3 | 70716 | 70255 | 70159 | 66945 | 60577 |
| 4 | 72849 | 72234 | 72247 | 69429 | 63361 |
| 5 | 75486 | 75051 | 75178 | 71834 | 65308 |
| 6 | 77601 | 77231 | 76765 | 73796 | 65649 |
| 7 | 80071 | 79905 | 79220 | 75254 | 67717 |
| 8 | 81579 | 82124 | 81520 | 78569 | 68391 |
| 9 | 82036 | 81738 | 80611 | 76082 | 68432 |
| 10 | 87161 | 86845 | 85602 | 84204 | 72859 |
| 11 | 89986 | 89607 | 88011 | 85695 | 76424 |
| 12 | 92258 | 91574 | 90904 | 87365 | 76098 |
| 13 | 93927 | 93803 | 93314 | 89635 | 77646 |
| 14 | 96607 | 96408 | 95287 | 92182 | 80259 |
| 15 | 98829 | 97837 | 97338 | 92505 | 81330 |
| 16 | 101618 | 100594 | 100365 | 94934 | 84565 |
| 17 | 102250 | 102273 | 100964 | 97933 | 88062 |
| 18 | 105171 | 105481 | 104358 | 99551 | 88296 |
| 19 | 107565 | 105643 | 107075 | 101610 | 88704 |
| 20 | 109100 | 108547 | 108960 | 104203 | 90617 |
| 21 | 111208 | 109235 | 110751 | 105674 | 91764 |
| 22 | 114717 | 112568 | 112262 | 107623 | 91833 |
| 23 | 114817 | 114364 | 114703 | 108488 | 95367 |
| 24 | 116482 | 116545 | 115588 | 109469 | 95605 |
| 25 | 118619 | 118375 | 117139 | 111877 | 99884 |
| 26 | 119590 | 120852 | 118571 | 115383 | 95577 |
| 27 | 122940 | 120587 | 120410 | 116405 | 99988 |
| 28 | 122051 | 122979 | 122237 | 118176 | 101475 |
| 29 | 126967 | 125672 | 123797 | 117745 | 102315 |
| 30 | 126876 | 126004 | 125378 | 118665 | 101640 |
| 31 | 125813 | 126005 | 126659 | 119992 | 102494 |
| 32 | 126616 | 128243 | 128788 | 126084 | 103133 |
| 33 | 128252 | 129417 | 130229 | 122053 | 103816 |
| 34 | 130757 | 128503 | 127232 | 123029 | 105060 |
| 35 | 132719 | 129127 | 131632 | 127140 | 102965 |
| 36 | 133015 | 132781 | 130355 | 125528 | 109004 |
| 37 | 134077 | 130211 | 131705 | 125563 | 106119 |
| 38 | 133186 | 134965 | 133731 | 128768 | 107968 |
| 39 | 135593 | 136313 | 133218 | 128257 | 108304 |
| 40 | 135800 | 135026 | 136224 | 128022 | 105141 |
| 41 | 135351 | 135906 | 134739 | 121753 | 104656 |
| 42 | 135313 | 135220 | 133547 | 132712 | 106827 |
| 43 | 132530 | 135929 | 135713 | 125680 | 108293 |
| 44 | 137685 | 134022 | 133618 | 128539 | 110736 |
| 45 | 130322 | 127367 | 131428 | 128979 | 106265 |
| 46 | 129679 | 124974 | 137632 | 126675 | 107821 |
| 47 | 130422 | 130387 | 136230 | 127905 | 110034 |
| 48 | 130933 | 129327 | 126266 | 125094 | 108614 |
| 49 | 127568 | 130004 | 127634 | 125812 | 106502 |
| max | 137685 | 136313 | 137632 | 132712 | 110736 |
| pos | 44 | 39 | 46 | 42 | 44 |

**Tabla 7.4**: Resultados del grupo de cocineros 3, ganancias en **€**.

| num orden | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pizzas | 4,30 | 4,40 | 4,50 | 4,60 | 4,70 | 4,80 | 4,90 | 5,00 | 5,10 | 5,20 | 5,30 | 5,40 | 5,50 | 5,60 | 5,70 | 5,80 | 5,90 | 6,00 | 6,10 | 6,2 |
| ensaladas | 3,40 | 3,50 | 3,60 | 3,70 | 3,80 | 3,90 | 4,00 | 4,10 | 4,20 | 5,30 | 4,40 | 4,50 | 4,60 | 4,70 | 4,80 | 4,90 | 5,00 | 5,10 | 5,20 | 5,30 |
| hamburguesas | 5,00 | 5,10 | 5,20 | 5,30 | 5,40 | 5,50 | 5,60 | 5,70 | 5,80 | 5,40 | 6,00 | 6,10 | 6,20 | 6,30 | 6,40 | 6,50 | 6,60 | 6,70 | 6,80 | 6,90 |
| precio medio | 4,49 | 4,59 | 4,69 | 4,79 | 4,89 | 5,00 | 5,10 | 5,20 | 5,3 | 5,33 | 5 | 5,60 | 5,71 | 5,81 | 5,91 | 6,01 | 6,11 | 6,21 | 6,31 | 6,41 |

**Tabla 7.5**: Valores ensayados de precios de cada menú y precio medio ponderado (20 primeros).

| num orden | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pizzas | 6,3 | 6,40 | 6,5 | 6,60 | 6,70 | 6,8 | 6,90 | 7 | 7,10 | 7,20 | 7,30 | 7,40 | 7,5 | 7,60 | 7,70 | 7,80 | 7,90 | 8,00 | 8,10 | 8,20 |
| ensaladas | 5,4 | 5,50 | 5,60 | 5,70 | 5,8 | 5,90 | 6,00 | 6,10 | 6,20 | 6,30 | 6,40 | 6,50 | 6,60 | 6,70 | 6 | 6,90 | 7,00 | 7,10 | 7,20 | 7 |
| hamburguesas | 7 | 7,10 | 7,20 | 7,30 | 7,40 | 7,5 | 7,6 | 7,7 | 7,8 | 7,90 | 8,00 | 8,10 | 8,20 | 8,30 | 8,40 | 8,50 | 8,60 | 8,70 | 8,80 | 8 |
| precio medio | 6,51 | 6,62 | 6,72 | 6,82 | 6,92 | 7,02 | 7,12 | 7,22 | 7,32 | 7,42 | 7,52 | 7,62 | 7,72 | 7,82 | 7,92 | 8,03 | 8,13 | 8,23 | 8,33 | 8 |

**Tabla 7.6**: Valores ensayados de precios de cada menú y precio medio ponderado (20 siguientes).

| num orden | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 |
|---|---|---|---|---|---|---|---|---|---|---|
| pizzas | 8,30 | 8,40 | 8,5 | 8,60 | 8,70 | 8,80 | 8,90 | 9 | 9,10 | 9 |
| ensaladas | 7,4 | 7,50 | 7,60 | 7,70 | 7,8 | 7,90 | 8,00 | 8,10 | 8,20 | 8,30 |
| hamburguesas | 9,00 | 9,10 | 9,20 | 9,3 | 9,40 | 9,50 | 9,60 | 9,70 | 9,8 | 9,90 |
| precio medio | 8,53 | 8,63 | 8,73 | 8,83 | 8,93 | 9,03 | 9,13 | 9,23 | 9,33 | 9,43 |

**Tabla 7.7**: Últimos 10 valores de los precios ensayados.

| num orden | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| mesas fijas | 0 | 5 | 10 | 15 | 20 |
| mesas configurables | 30 | 25 | 20 | 15 | 10 |

**Tabla 7.8**: Combinaciones de mesas probadas.

| planes de cocineros | | | | |
|---|---|---|---|---|
| num | 0 | 1 | 2 | 3 |
| 0-1h | 0 | 0 | 0 | 0 |
| 1-2h | 0 | 0 | 0 | 0 |
| 2-3h | 0 | 0 | 0 | 0 |
| 3-4h | 0 | 0 | 0 | 0 |
| 4-5h | 0 | 0 | 0 | 0 |
| 5-6h | 0 | 0 | 0 | 0 |
| 6-7h | 0 | 0 | 0 | 0 |
| 7-8h | 0 | 0 | 0 | 0 |
| 8-9h | 0 | 0 | 0 | 0 |
| 9-10h | 0 | 0 | 0 | 0 |
| 10-11h | 0 | 0 | 0 | 0 |
| 11-12h | 10 | 10 | 5 | 5 |
| 12-13h | 15 | 15 | 10 | 5 |
| 13-14h | 35 | 35 | 30 | 25 |
| 14-15h | 40 | 35 | 35 | 30 |
| 15-16h | 15 | 15 | 10 | 5 |
| 16-17h | 20 | 15 | 10 | 10 |
| 17-18h | 35 | 30 | 30 | 30 |
| 18-19h | 40 | 35 | 35 | 30 |
| 19-20h | 35 | 30 | 25 | 20 |
| 20-21h | 35 | 30 | 25 | 20 |
| 21-22h | 20 | 15 | 10 | 10 |
| 22-23h | 15 | 10 | 10 | 5 |
| 23-24h | 5 | 5 | 5 | 5 |

**Tabla 7.9**: Planes de cocineros probados.

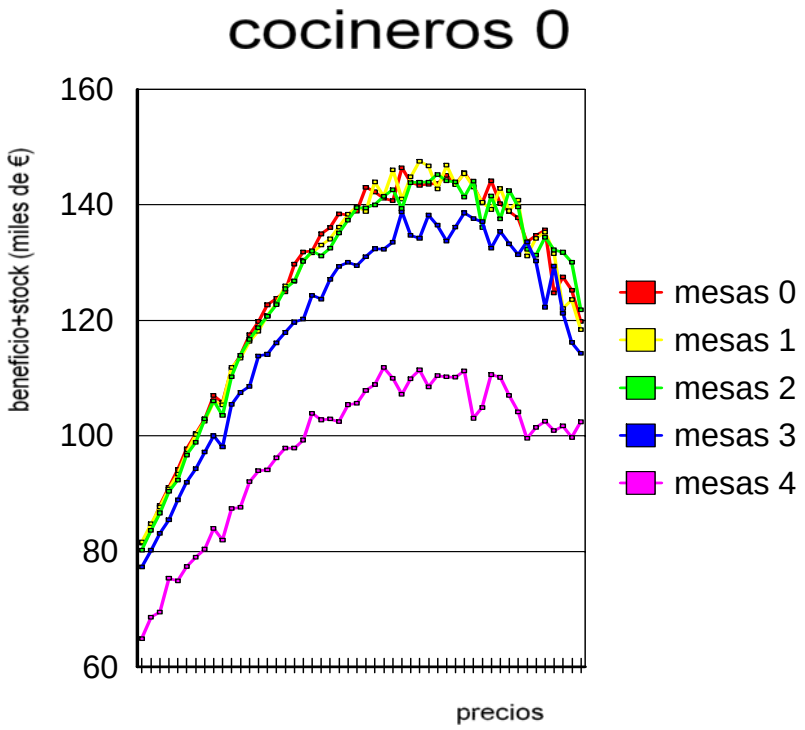


**Fig. 7.1**: Gráfica de beneficio+stock frente a precios para el grupo de cocineros 0.

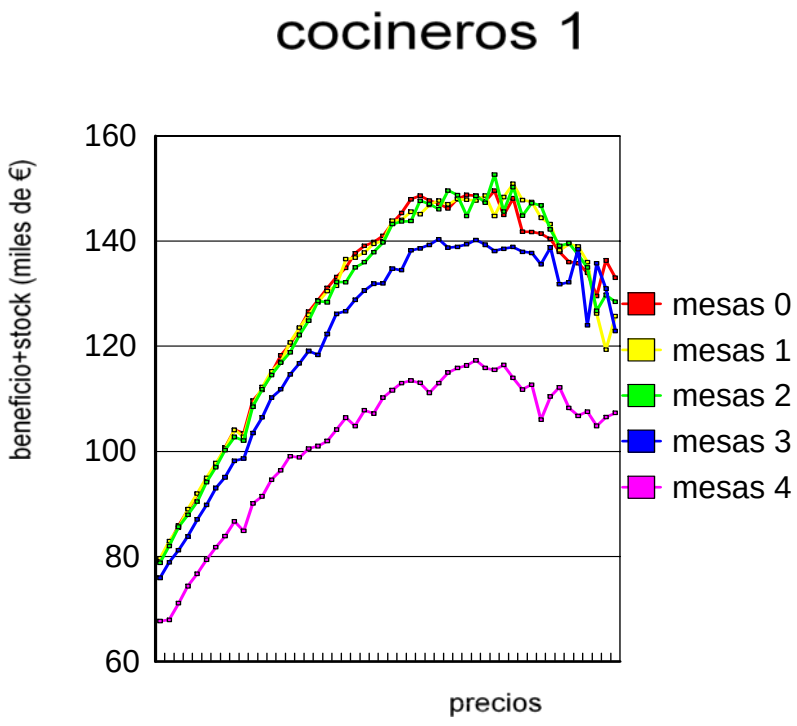


**Fig. 7.2**: Gráfica de beneficio+stock frente a precios para el grupo de cocineros 1.

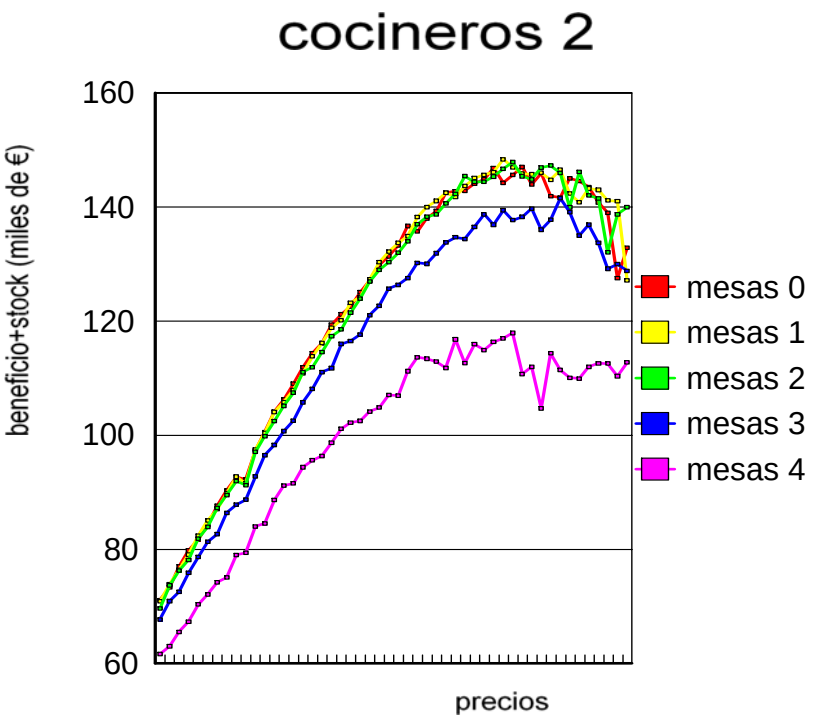


**Fig. 7.4**: Gráfica de beneficio+stock frente a precios para grupo de cocineros 2.

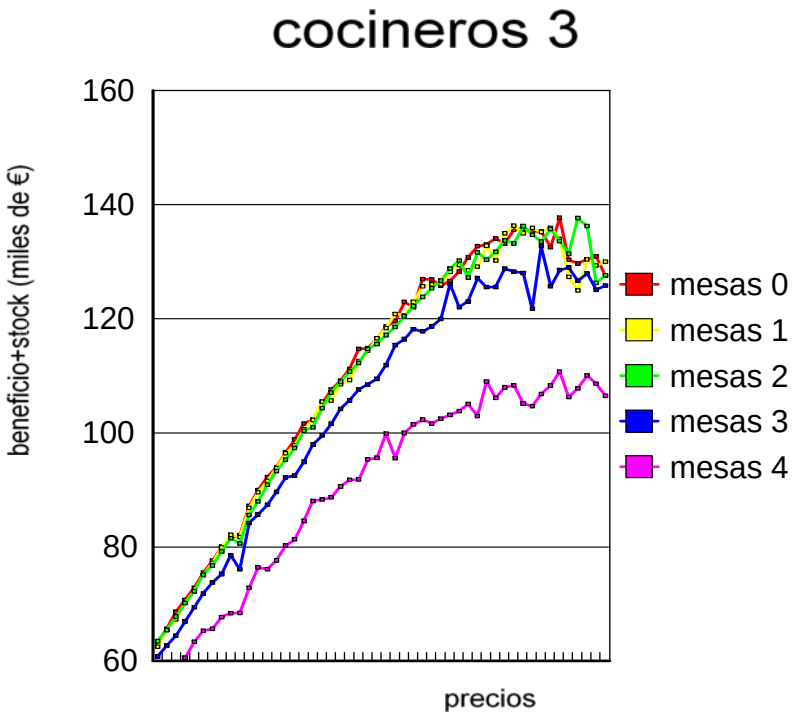


**Fig. 7.3**: Gráfica de beneficio+stock frente a precios para grupo de cocineros 3.

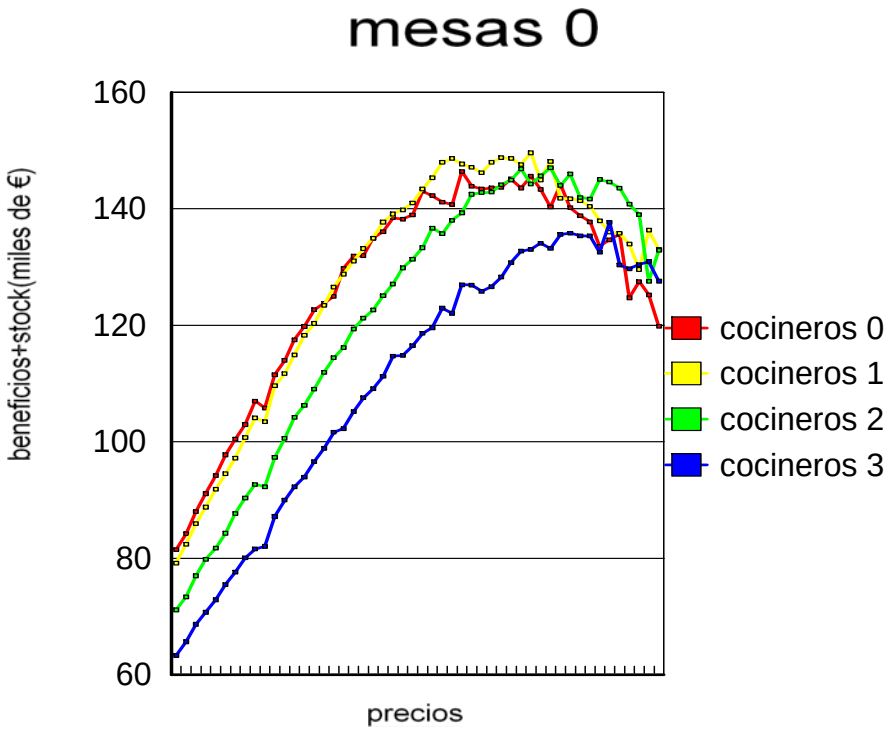


**Fig. 7.5**: Gráfica de beneficio+stock frente a precios para el grupo de mesas 0.

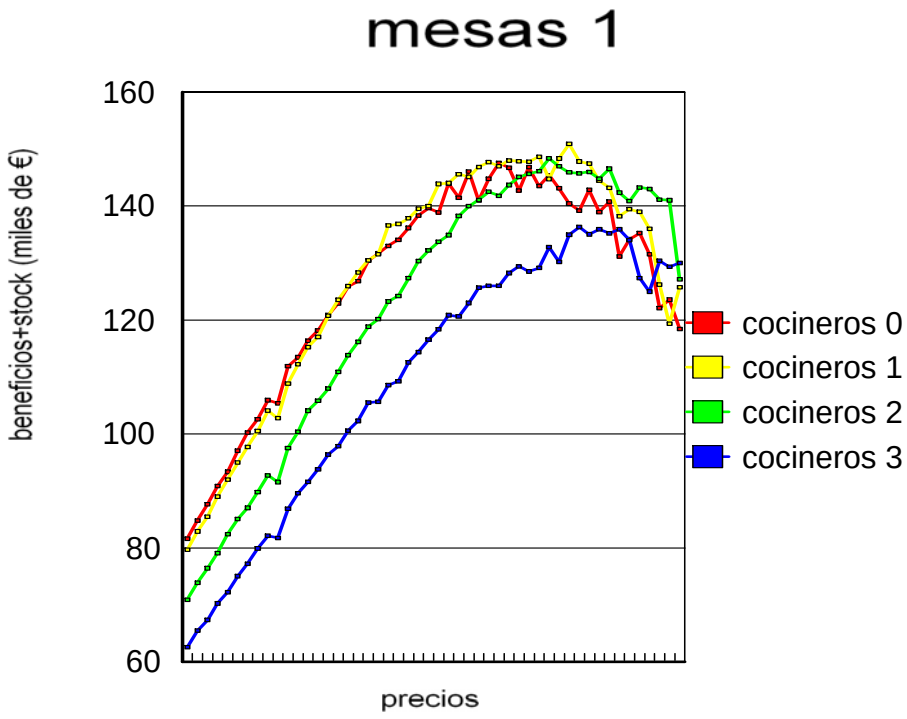


**Fig. 7.6**: Gráfica de beneficio+stock frente a precios para grupo de mesas 1.

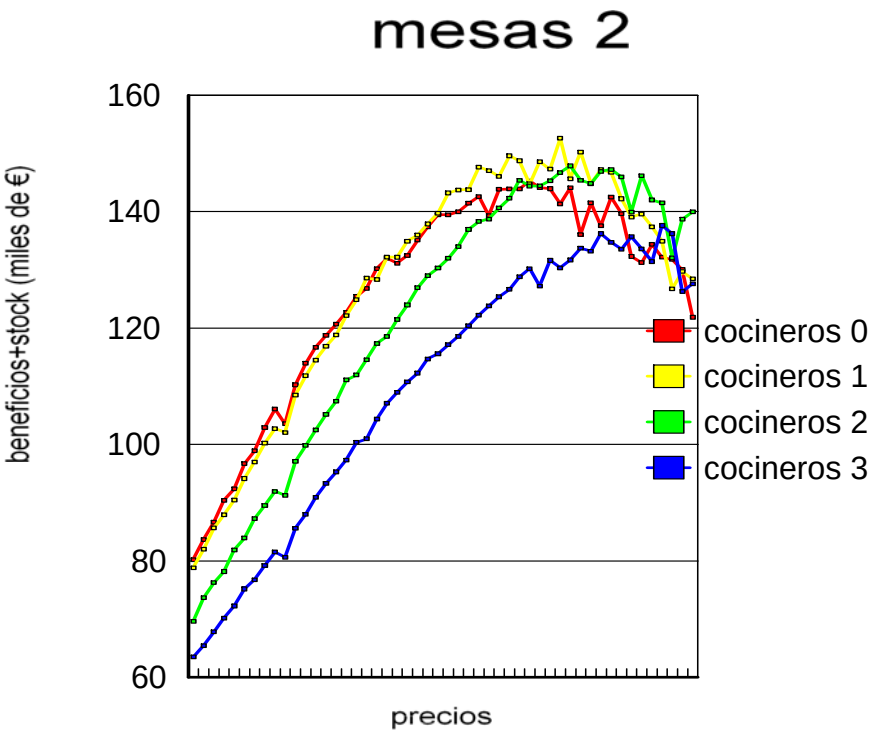


**Fig. 7.7**: Gráfica de beneficio+stock frente a precios para grupo de mesas 2.

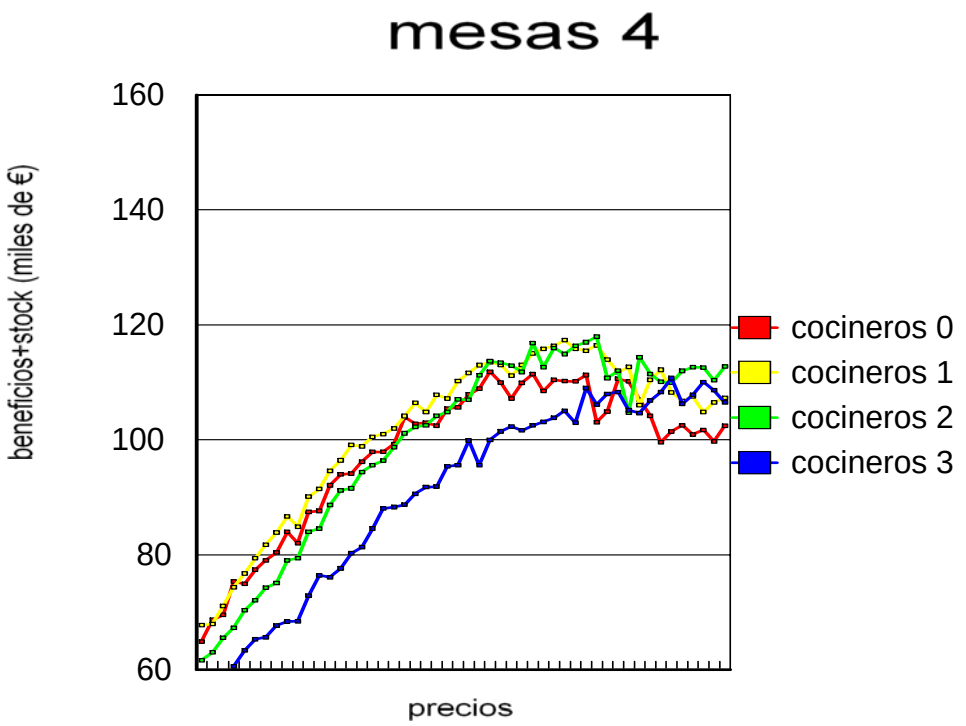


**Fig. 7.9**: Gráfica de beneficios+stock para grupo de mesas 4.

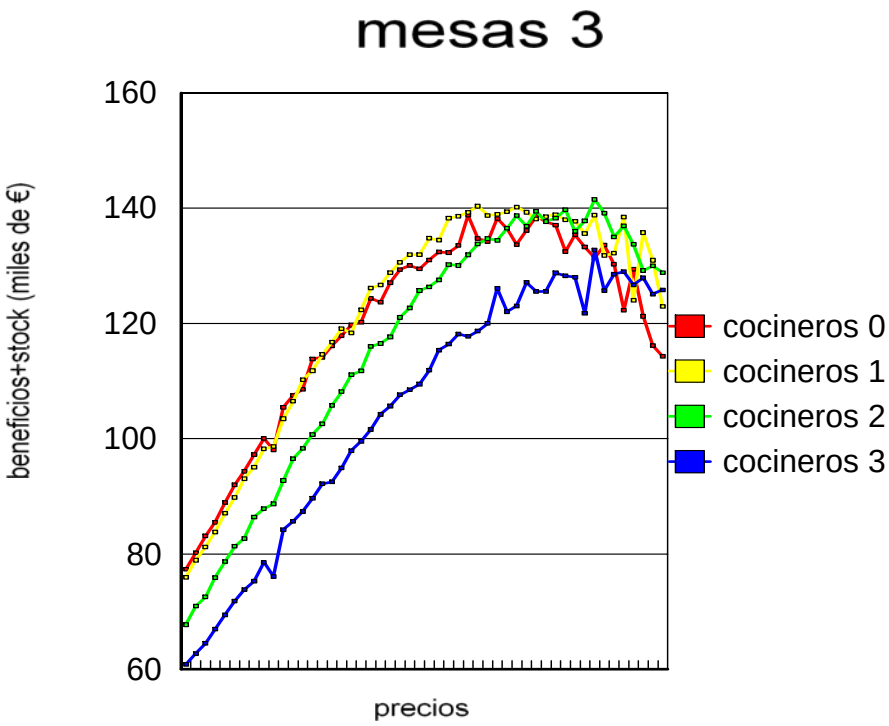


**Fig. 7.8**: Gráfica de beneficio+stock frente a precios para grupo de mesas 3.

| | Arena 7.0 | | | Simkit | | |
|---|---|---|---|---|---|---|
| | cocineros 1 | | | cocineros 1 | | |
| | m0 | m1 | m2 | m0 | m1 | m2 |
| 32 | | | 146384 | 147683 | 146865 | 147050 |
| 33 | | | 149220 | 147084 | 147704 | 146058 |
| 34 | 148347 | 148832 | 147657 | 146195 | 146983 | 149614 |
| 35 | 148789 | 148370 | 149710 | 147958 | 147990 | 148743 |
| 36 | 147960 | 149478 | 147546 | 148767 | 147855 | 144777 |
| 37 | 151549 | 147215 | 146368 | 148627 | 147752 | 148604 |
| 38 | 145058 | 146728 | 147030 | 147607 | 148655 | 147297 |
| 39 | 145421 | | 144526 | 149604 | 144750 | 152627 |

**Tabla 7.10**: Comparación de resultados de Arena y Simkit en zona de máximos.

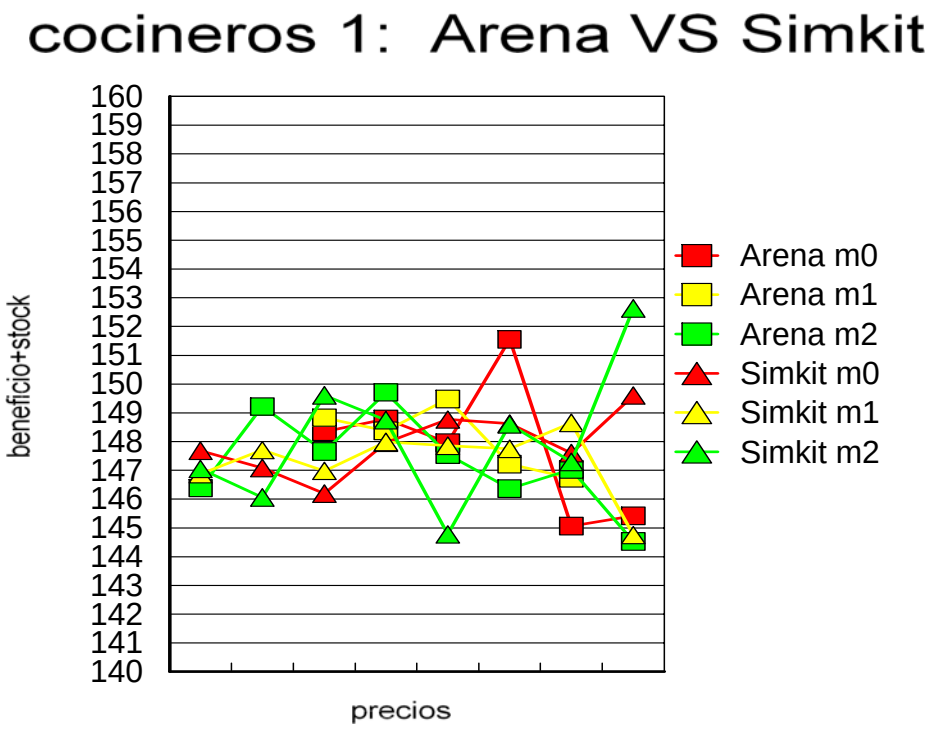


**Fig. 7.10**: Gráfica en la que se comparan los resultados máximos de Arena y Simkit.

## 7.4.-Conclusiones

Los resultados de esta simulación más amplia reafirma las conclusiones obtenidas en el Capítulo 5 con Arena: Escasa sensibilidad a la relación numero de mesas fijas/configurables siempre que el número de mesas fijas se mantenga por debajo de 10.

Combinaciones con mayor cantidad de personal alcanza sus máximo a precios menores que aquellas que tienen menos cantidad de personal etc.

Los resultados obtenidos con las mismas combinaciones de datos de entrada dan resultados equivalentes en ambas implementaciones (Arena y Simkit). La simulación en Simkit alcanza el valor óptimo de 152627€ de beneficios con 7 grupos de 5 cocineros, con 10 mesas fijas y 20 configurables y a un precio medio de 8,13€ (similar al de 8,23€ de Arena, que también se da con 7 grupos de 5 cocineros, y un valor óptimo de 151549€).

Fuera de la coincidencia de los resultados obtenidos, se ha demostrado la capacidad de las librerías Simkit para poder desarrollar en un tiempo razonable modelos complejos.

# Capítulo 8.-Conclusiones y líneas de trabajo futuras

## 8.1.-Conclusiones

Se ha apreciado la facilidad y potencia del diseño usando los diagramas de gráficos de eventos, dentro del paradigma orientado a eventos.

Las librerías de Simkit aportan un apoyo básico para implementar modelos en un lenguaje de propósito general como es Java, con la flexibilidad que esto conlleva. Este apoyo se centra en la gestión de la lista de eventos futuros, los métodos para implementar las distintas clases de eventos y la revocación de los mismos, gestión del tiempo de simulación; y objetos para el manejo de las distribuciones estadísticas genéricas.

## 8.2.-Líneas de trabajo futuras

Se echa en falta procedimientos genéricos para la implementación del equivalente a “entidades”, “recursos” ,“colas” etc. en el modelo orientado al proceso, librerías de funciones estadísticas más amplias que contemplen distribuciones más particulares como distribuciones exponenciales con parámetro variable de forma periódica etc., así como modelos genéricos para la interacción gráfica entre el usuario y el modelo en proceso de creación.

Se puede contemplar como futuras tareas la creación de “interfaces” o “clases abstractas” Java que recojan las propiedades y métodos generales necesarios para la gestión de elementos típicos

de la simulación de procesos como son los ya comentados: “entidades”, “recursos”, colas etc. En el modelo simple se han incluido clases abstractas para modelar recursos que incluyen sus propias colas, una clase modelo para entidades. Utilizando estas clases en el modelo simple se simplifica de forma importante la creación de recursos y entidades.

En el desarrollo del modelo del establecimiento de comida rápida se ha visto que para modelar los “recursos” parece conveniente:

que el método que produce su ocupación que devuelva un “double” positivo que corresponda al tiempo de servicio de dicho recurso salvo que el recurso no esté disponible que se devuelve -1;

la gestión del tiempo de ocupación que se puede resolver mediante una variable que acumula el tiempo total de ocupación del recurso, para aquellos que han sido ocupados y liberados. Cuando se ocupa un recurso se incluye una referencia del instante de tiempo en una LinkedList. Cuando se libere un recurso se saca la primera referencia de dicha `LinkedList` y la diferencia entre el tiempo de simulación y la incluida en la referencia se suma al tiempo global de ocupación. Cuando finalice la simulación se obtienen todas las referencias que todavía quedan en la `LinkedList`, se calcula el la diferencia entre el tiempo final y el tiempo en que se incluyeron en la `LinkedList` y se acumula al tiempo de ocupación. Este mecanismo es suficiente si el tiempo de ocupación se calcula globalmente sobre el conjunto de los recursos de un tipo, que es la situación de los modelos del proyecto.

Si se quieren calcular estadísticas de ocupación en que se tenga que conocer datos concretos de cada unidad de recurso el mecanismo anterior no es suficiente. Sería necesario identificar con una clave cada unidad de recurso. La `LinkedList` anterior no sería suficiente puesto que en ella no se guardan referencias del recurso concreto ocupado. Aunque se guardaran referencias no

sería conveniente dicha estructura de datos ya que la búsqueda en ella de un recurso individual no sería eficaz [Bran00]. En lugar de la `LinkedList` se podría usar un `Map` o una `Hashtable`. Se cargaría la referencia temporal del recurso individual usando como clave su identificador. Para cada recurso individual se guardaría una variable que acumulara su tiempo global de ocupación. Cuado se ocupara se cargaría en el `Map` el instante de tiempo en que se ocupó y al liberarse se descargaría del `Map`, se calcularía la diferencia entre el instante de entrada y de salida, y ese tiempo se acumularía en la variable que acumulara el tiempo de dicho recurso. No ha sido necesario este método para ninguno de los modelos desarrollados en este Proyecto.

El máximo número de recurso ocupados en algún instante se puede gestionar mediante una variable que registre dicho valor. Se gestiona de forma fácil. Cuando se solicita una nueva ocupación si tiene éxito, se comparan los ocupados con el máximo hasta ese momento, si es mayor el número de ocupado se actualiza la variable con el valor de los ocupados.

El método que libera los “recursos” , que además de dejar los “recursos” de nuevo a disposición, debería actualizar las variables correspondientes a las estadísticas.

En cuanto a las colas, se han implementado mediante `LinkedList`, cada elemento nuevo que entra se coloca al final. Cuando hay recurso disponible se retira el primero. El “balking” se implementa fácilmente mediante controlando el tamaño de la `LinkedList`. En el modelo del establecimiento de comida rápida los tiempos de permanencia en cola, en la simulación se han controlado mediante propiedades de la “entidades” encoladas. Se ha visto que es más general (ya que no tiene por que alterarse una propiedad de la entidad) incluir en la `LinkedList` en lugar de la entidad un objeto formado por la entidad y por el tiempo en que se puso en la cola. Este segundo mecanismo se ha usado en las colas del modelo simple ya que se desarrollo después, además se han incluido como parte de los objetos que representan a los recursos.

Si en la cola se da la posibilidad de abandonar, es preciso poder localizar la “entidad” a retirar, en nuestro modelo se ha resuelto mediante los métodos normales de `LinkedList`, que permite localizar en la lista un objeto por su valor, la búsqueda comenzará por el primero y finalizará por el último lo que da una complejidad algorítmica proporcional al tamaño de la cola. Esta complejidad [Lewi06] no es problemática si se cree que la cola no será muy larga (en nuestro modelo no puede ser mayor que 5), si se sospecha que alcance valores largos, se puede conseguir una mejor eficacia si la `LinkedList` se combina con un `Map`. La búsqueda de la entidad en el `map` se podría hacer en un sólo paso, y se retiraría dicho elemento de las dos estructuras.

Las “entidades”, en general deberían tener una clave de identificación, en los modelo se les ha asignado un entero de tipo `long`. También es conveniente que guarden el tiempo de simulación en que fueron creadas. Para generar la identificación de forma única, se ha guardado una variable estática de tipo `long` en la clase de creación, cada vez que se crea una se le asigna el valor de dicha variable como identificador y se incrementa dicho valor para la siguiente, de esta manera además puede informar del número de entidades creadas.

Se ha visto que es conveniente poder seguir la pista a las entidades creadas en la fase de desarrollo del modelo, por lo que se ha creado un `Map` en el que usando como clave el número de identificación se guarde el objeto que represente a la “entidad”. Cuando las entidades no son necesarias (en Arena sería al llegar a módulos “dispose”) se retiran del `Map` por lo que quedarían las que todavía están en juego y se puede obtener información de ellas que nos puede ayudar a afinar el modelo.

Para el modelo desarrollado ha sido necesario implementar una función que recibiendo una planificación diaria que se repetiría a lo largo del tiempo proporcione el retraso exponencial con

los parámetros variables que corresponden a esa planificación. Se ha creado la clase `ExpoParVarPer` que tiene esa característica.

Como el modelo simple se ha creado con posterioridad al desarrollo del establecimiento de comida rápida, se han creado modelos para las entidades y recurso. En dichos modelos se han recogido algunas de las necesidades localizadas cuando se desarrolló el establecimiento de comida rápida. Extendiendo las clases modelos de recursos y entidades se pueden crear los necesarios para el modelo introduciendo muy poco código Java. Como tarea futura quedaría mejorar los modelos y detectar nuevas necesidades.

## Bibliografía


[Bank96] Banks, J., Carson J. S. , Nelson, B. L. (1996), *"Discrete-Event System Simulation"*, Prentice Hall.

[Box 78] Box, G. E. P., Hunter, J. S. (1978) *"Statistics for Experimenters: an Introduction to Design, Data Analysis and Model Building"*, John Wiley & Sons.

[Bran00] Branssard, G., Bratley, P., 2000, *"Fundamentos de Algoritmia"*, Prentice Hall.

[Brat87] Bratley, P., Fox, B. L., Schrage, L. E. (1987), *"A Guide to Simulation"*, Springer.

[Buss01a] Buss, A., 2001, *"Technical Notes, Basic Event Graph Modeling"*, Operations Research Department, Naval Postgraduate School Monterey, CA 93943-5000 U.S.A.

[Buss01b] Buss, A., 2001, *"Technical Notes, Discrete Event Programing With Simkit"*, Operations Research Department, Naval Postgraduate School Monterey, CA 93943-5000 U.S.A.

[Caho06] Cahoon, J.(2006), *"Programación en Java 5.0"*, McGraw-Hill/Interamericana de España, S.A.

[Cass99] Cassandras, C. G., Laforntune, S. (1999), *"Introduction to Discrete Event Systems"*, Kluwer Academic Publishers.

[Ceba06] Ceballos, Francisco Javier de , 2006, "*Enciclopedia de Microsoft Visual Basic"*, Ra-Ma

[Fish78] Fishman G. S.(1978), *"Principles of Discrete Event Simulation"*, John Wiley & Sons.

[Frou05] Froufe Quintas, A., (2005), *"Java 2: Manual de Usuario y Tutorial"*, Ra-Ma.

[Guas02] Guasch, A., Piera, M. A., Casanovas, J., Figueras, J. (2002), *"Modelado y Simulación. Aplicación a Procesos Logísticos de Fabricación y Servicios"*, Editions UPC.

[Kelt02] Kelton, W. D., Sadowski, R. P., Sadowski,D. A., 2002, *"Simulation Whith Arena"*, 2ª edición, Mc Graw Hill.

[Lewi06] LEWIS, J., CHASE, J.,(2006), *"Estructura de Datos con Java"*, Pearson Education.

[Ljun94] Ljung, L., Torkel, G. (1994), *"Modeling of Dynamic Systems"*, Prentice-Hall.

[Papo91] Papoulis, A. (1991), *"Probability, Random Variables and Stochastic Processes"*, McGraw-Hill.

[Sier82] Sierra Andrés, F. de la, 1982, *"Economía"*, E. T. S. de Ingeniería Industrial, UNED.

[Urqu06a] Urquía, A., 2006, *"Simulación, Texto Base de Teoría"*. Departamento de Informática y Automática, E. T. S. de Ingeniería Informática, UNED.

[Urqu06b] Urquía, A., 2006, *"Simulación, Solución a una Serie de Problemas, Enunciado del Proyecto y Exámenes de Anteriores Convocatorias "*. Departamento de Informática y Automática, E. T. S. de Ingeniería Informática, UNED.