



HORA 20

Orientación a las interacciones y cambios de estado

A continuación profundizará en los casos de uso analizados en la hora anterior. Los utilizará como fundamento para comprender las partes del sistema y la forma en que se comunican entre sí.

En esta hora se tratarán los siguientes temas:

- Las partes funcionales del sistema
- Modificación de los casos de uso
- Análisis de la colaboración entre las partes funcionales

El análisis de los casos de uso de la hora anterior es un gran avance para hacer realidad el sistema RICO. No obstante, el análisis aún está muy lejos de permitir que se empiece con la codificación.

Analizar los casos de uso ha ayudado a visualizar las partes funcionales del sistema. Aunque ahora sabemos mucho de ellos, aún tenemos que modelar la forma en que las partes funcionales se comunicarán entre sí y la forma (y momento) en que cambiarán de estado. Dar esta información a los programadores facilitará su trabajo. Tendrán una visión de la forma de codificar las clases y hacer que trabajen en conjunto.

Las partes funcionales del sistema

Una forma de empezar es enumerar los componentes sugeridos del sistema en cada paquete de casos de uso. Aunque no hemos analizado de manera explícita todos los casos de uso en todos los paquetes en la hora anterior, aún podríamos obtener los componentes que se asumen en ellos. Claro que en un verdadero proyecto de desarrollo, un equipo de desarrollo debería analizar todos los casos de uso antes de continuar.

El paquete Mesero

Al finalizar la hora anterior, enumeramos las partes de software del sistema de acuerdo con nuestro análisis de los primeros nueve casos de uso del paquete Mesero: en las palm-tops, RICO necesitará interfaces para el registro de órdenes, su modificación, su control, el estado de los clientes y el envío de mensajes. También será necesaria una pantalla principal en la interfaz. Nuestro análisis trajo a la luz la necesidad de una interfaz para ver el avance de los pedidos en la PC de la cocina. RICO necesitará una base de datos para contener a todas las órdenes.

Los casos de uso que no analizamos también sugieren componentes del sistema. Para refrescar su memoria, esos casos eran:

- Llamar a un mozo de piso
- Tomar una orden de bebida
- Transmitir una orden de bebida al bar
- Obtener un acuse de recibo
- Recibir una notificación del bar

Los casos de uso sugieren algunos componentes evidentes. El primero nos indica que algo en la interfaz del Mesero (como una pantalla dedicada) tiene que permitir la llamada a un mozo de piso. El segundo, que se necesita una pantalla para obtener una orden de bebidas (similar a la que se usa para tomar una orden de alimentos). La interfaz debe poder obtener un acuse de recibo (para mostrar, por decir, que el mozo de piso ha recibido la petición) y recibir un mensaje del bar que indique que está lista la bebida.

Debido al trabajo de un mesero, no debe sorprendernos que los componentes primordiales en este paquete son interfaces relacionadas con el registro de órdenes, así como la recepción y envío de mensajes.

El paquete Chef

Los casos de uso del paquete Chef son:

- Almacenar una receta
- Obtener una receta
- Notificar al mesero
- Recibir una petición del mesero
- Dar acuse de recibo a una petición del mesero
- Indicar el periodo de preparación
- Asignar una orden

¿Qué componentes sugieren estos casos de uso? Nuevamente, algunos vienen a la mente de manera evidente.

El paquete Mozo De Piso

Los casos de uso para el mozo de piso son:

- Recibir una petición del mesero
- Dar acuse de recibo a una petición
- Indicar que una mesa ha sido limpiada

El paquete Asistente Mesero

Como recordará, decidimos dividir el paquete Asistente en Asistente Mesero y Asistente Chef. Los casos de uso del Asistente Mesero serían:

- Obtener una petición del mesero
- Dar acuse de recibo a la petición
- Notificar que se ha completado la petición

El paquete Asistente Chef

Los casos de uso del Asistente Chef serían:

- Obtener una petición del chef
- Dar acuse de recibo a la petición
- Notificar que se ha completado la petición

Tal vez podría pensarse que no es necesaria una computadora para un asistente de chef dado que trabaja de forma muy cercana al chef en la cocina. No obstante, si la cocina es muy grande, la comunicación electrónica podría ser una buena idea.

El paquete Cantinero

Los casos de uso del Cantinero son:

- Capturar la receta de una bebida
- Obtener la receta de una bebida
- Recibir una notificación del mesero
- Recibir una petición del mesero
- Dar acuse de recibo de una petición
- Notificar que la petición se ha completado

Estos casos de uso son similares a los del paquete del Chef, y los componentes de software que sugieren también lo son. El hardware es similar, a su vez: detrás de una barra, una PC de escritorio podría tener más sentido que una palmtop.

Necesitamos una base de datos de recetas para bebidas y una interfaz que permitan un fácil acceso a ella (tanto para capturar como para obtener recetas). La interfaz para el cantinero tiene que mostrar una notificación proveniente del mesero (de que la mesa de un cliente está lista) y una petición de una bebida también proveniente del mesero. El cantinero tendrá que enviar un acuse de recibo de que la petición se ha recibido y, también, notificar al mesero que la bebida está lista.

El paquete Encargado Del Guardarropa

Los casos de uso del Encargado Del Guardarropa son:

- Imprimir un vale de abrigo
- Imprimir un vale de sombrero

Los componentes de software en la palmtop del encargado del guardarropa deberían incluir una interfaz que permita la impresión del vale adecuado. El vale deberá incluir la hora y una descripción del artículo. Posiblemente también queramos tener una base de datos de elementos almacenados.

Colaboración en el sistema

En este punto del proyecto, la tarea será mostrar la forma en que los componentes interactúan para completar cada caso de uso. Modelaremos tales interacciones para un par de casos de uso en el paquete del Mesero. El conjunto de casos de uso es demasiado grande para que los veamos todos; pero en un proyecto real, un equipo de desarrollo hará justamente eso.



Recuerde que, como dije antes: Detrás de cada caso de uso se esconde un diagrama de interacciones.

Tomar una orden

Empecemos con el caso de uso “Tomar una orden”. De lo desprendido en la hora 19, los pasos son:

1. En la palmtop, el Mesero activa la interfaz para capturar una orden.
2. Aparece la interfaz para capturar órdenes.
3. El Mesero registra la selección del Cliente hecha del menú en RICO.
4. El sistema transmite la orden a la PC de la cocina.

En el modelo que desarrollamos en la hora anterior, este caso de uso incluye a “Transmitir la orden a la cocina”, cuyos pasos son:

1. Hacer clic en un botón en la interfaz para el registro de órdenes indica “Enviar a la cocina”.
2. RICO transmitirá la orden mediante la red inalámbrica.
3. La orden llega a la cocina.
4. La interfaz del usuario para registrar las órdenes en la palmtop indica que la orden ha llegado a la cocina.

Un diagrama de secuencias podría mostrar bastante bien esta colaboración (igual lo haría un diagrama de colaboraciones, mismo que le pediré que genere en el ejercicio 1). La preparación del diagrama nos fuerza a enfocar nuestra imaginación de varias formas.

Para empezar, cuando el mesero toma la orden de un cliente, creará algo: ¡una orden! Tal orden es un objeto en el sistema RICO (a su vez, es una instancia de una clase, Orden, proveniente de nuestro análisis del dominio en la hora 17, “Elaboración de un análisis del dominio”). El chef la usará como orientación para iniciar y realizar un conjunto de acciones. El mesero totalizará una cuenta de acuerdo con ella. El cliente pagará la cuenta. Así, esta orden generada es un elemento importante.

A su vez, si examina los casos de uso “Cambiar una orden” y “Sondear el progreso de la orden” (como lo haremos en un momento), verá que se hace referencia a una lista de órdenes. Esta lista se habrá obtenido de una base de datos de órdenes, misma que mencioné al finalizar la hora 19. ¿Cómo se captura la orden en la base de datos? Bien, pues ello deberá ocurrir en este caso de uso.

Podemos enfocar nuestra imaginación de otra forma. En el caso de uso incluido, el término “cocina” es un poco vago. Dado que estamos modelando componentes de software, tendremos que pulir lo que queremos decir aquí. Imaginar cómo podría funcionar todo nos llevará a una forma donde el sentido común establecerá que la orden deberá mostrarse de algún modo en la interfaz del chef en la PC de la cocina. El cómo ocurra no es algo que en este momento deberá ocuparnos.

Luego de que hemos pensado en lo anterior, el caso de uso “Tomar una orden” luciría así:

1. En la palmtop, el Mesero activa la interfaz para capturar una orden.
2. Aparece la interfaz para capturar órdenes.
3. El Mesero registra en RICO la selección del Cliente hecha del menú.
4. RICO genera una orden.
5. Incluir(Transmitir una orden a la cocina).
6. El sistema guarda la orden en la base de datos de órdenes.



“RICO” y “el sistema” son sinónimos.

Observe el recurso utilizado en el paso 5 que indica la inclusión del caso de uso “Transmitir una orden a la cocina”.

La “base de datos de pedidos” emplaza nuestra imaginación hacia el lado del hardware. Esta base de datos debe encontrarse en una computadora, pero no hemos establecido una. Una posibilidad sería contar con un equipo de cómputo central que tuviera esta base de datos y la hiciera accesible a las demás máquinas de la red.

Mientras estamos en esto, debemos expandir el caso de uso “Transmitir la orden a la cocina” e incluir la modificación respecto a la interfaz del chef. Dado que tiene un paso que establece un mensaje en la palmtop del mesero que indica la recepción de la orden en la cocina, debemos agregar un paso que haga que el sistema envíe un mensaje de la PC de la cocina a la palmtop. Así, los pasos serían:

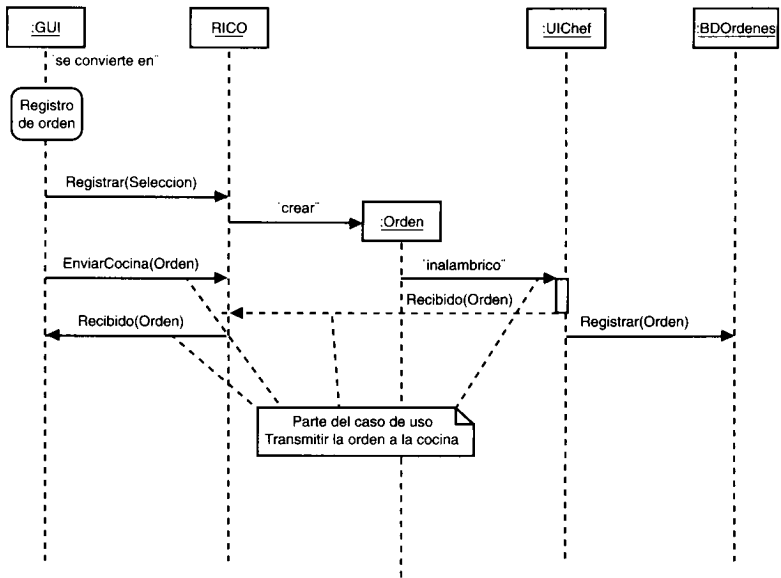
1. Hacer clic en un botón en la interfaz para el registro de órdenes indica “Enviar a la cocina”.
2. RICO transmitirá la orden mediante la red inalámbrica.
3. La orden llegará a la interfaz del chef en la PC de la cocina.
4. RICO enviará un mensaje de la cocina a la interfaz de la palmtop que acuse el recibo de la orden.
5. La interfaz del usuario, para registrar las órdenes en la palmtop, indicará que la orden ha llegado a la cocina.

Estas modificaciones a los casos de uso son sólo dos ejemplos más de la forma en que una fase de un proyecto puede influenciar a otra. En este contexto, la preparación de nuestros diagramas de secuencias nos ayudaron a pulir nuestra idea de los casos de uso con la base de esos diagramas.

La figura 20.1 le muestra el diagrama de secuencias que captura nuestra idea de este caso de uso. Para recordar lo que vio de los diagramas de secuencias, los objetos que están colocados en la parte superior de él representan a los componentes en este caso de uso. El objeto Orden se genera durante el caso de uso, y por ello está bajo los otros dos. El mensaje que apunta hacia él tiene un estereotipo «crear». La línea punteada que parte de cada objeto representa la “línea de vida de un objeto”, misma que avanza hacia abajo verticalmente. Los pequeños rectángulos en los tiempos de actividad se llaman “activaciones”. Cada activación representa al periodo en el que un objeto realiza una acción.

FIGURA 20.1

El diagrama de secuencias de “Tomar una orden”.



Vea el cambio de estado en el primer tiempo de actividad. Intenta aclarar la forma en que una interfaz maneja un tipo especial de actividad. Podríamos incluir todos los cambios de estado posibles como diagramas de estados por separado, pero sería excesivo. El colocarlos en diagramas de secuencias (al menos en este dominio) aparentemente es más sencillo.



En un diagrama de secuencias, una flecha de mensaje con una línea punteada representa a un mensaje devuelto.

Cambiar una orden

Intentemos con otra. En la hora anterior, los pasos de caso de uso “Cambiar una orden” fueron:

1. En la palmtop, el mesero activa la interfaz del usuario para cambiar una orden.
2. La interfaz del usuario trae una lista de órdenes realizadas y enviadas a la cocina por el mesero.
3. El mesero selecciona la orden por cambiar.
4. El mesero registra la modificación a la orden.
5. El sistema transmite la orden a la PC de la cocina.

Nuevamente, la preparación del diagrama nos ayuda a pulir nuestra imaginación y modificar de cierto modo al caso de uso. Luego del paso 4, sin duda queremos que el sistema cree una orden modificada. Luego del paso 5, el sistema debería registrar la orden modificada en la base de datos de órdenes.

Con ello, el nuevo caso de uso debería ser:

1. En la palmtop, el mesero activa la interfaz del usuario para cambiar una orden.
2. La interfaz del usuario trae una lista de órdenes realizadas y enviadas a la cocina por el mesero.
3. El mesero selecciona la orden por cambiar.
4. El mesero registra la modificación a la orden.
5. RICO crea una orden de acuerdo con la modificación a la anterior.
6. Incluir(Transmitir una orden a la PC de la cocina).
7. El sistema guarda la orden modificada en la base de datos de órdenes.

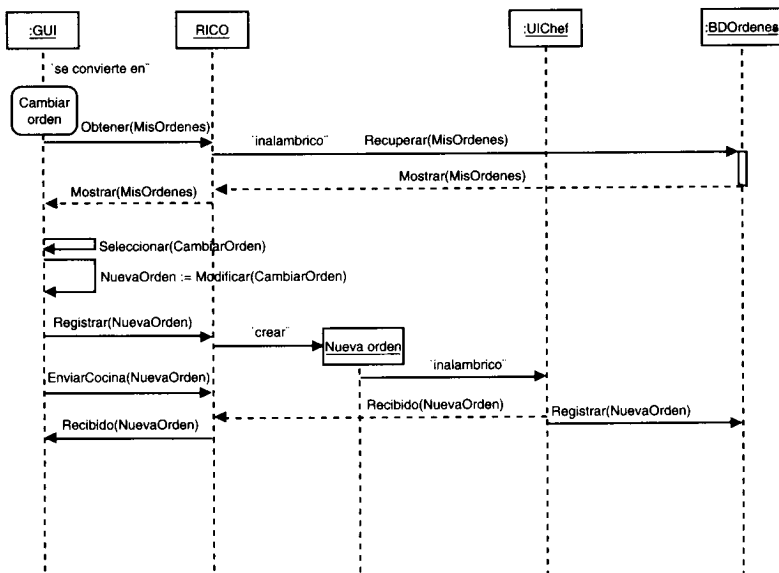
Nuevamente, utilizamos el recurso para indicar la inclusión de un caso de uso.

La figura 20.2 le muestra el diagrama de secuencias que corresponde a este caso de uso.

Al igual que en la figura 20.1, ahora mostramos un cambio de estado.

FIGURA 20.2

El diagrama de secuencias para “Cambiar una orden”.



Sondeo del progreso de la orden

Veamos otro caso de uso antes de terminar. El caso de uso “Sondear el progreso de la orden” consta de los siguientes pasos:

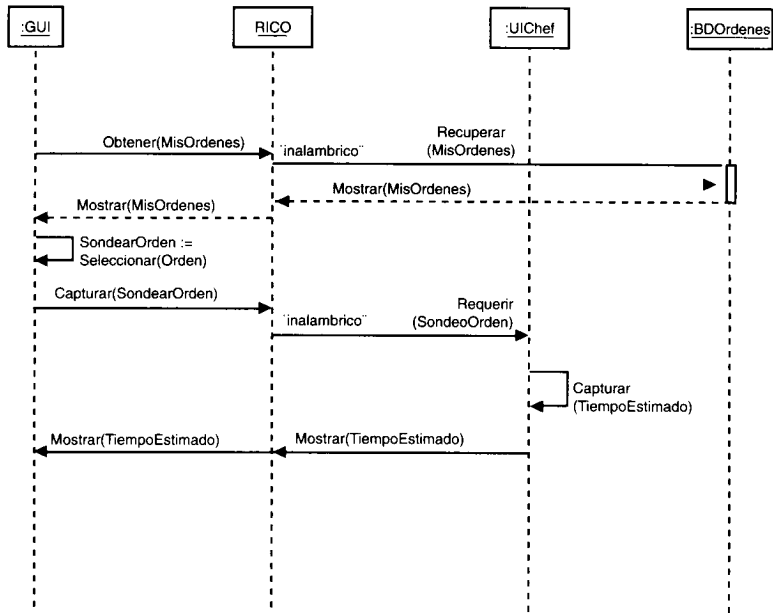
1. El mesero activa la interfaz en la palmtop para sondear una orden registrada.
2. La interfaz le muestra al mesero una lista de las órdenes que tiene registradas en la cocina.
3. El mesero elige la orden que desea sondear.
4. El sistema transmite un mensaje de sondeo a la PC de la cocina.
5. La PC de la cocina recibe el mensaje.
6. El chef selecciona la orden de la cual se quiere conocer su avance.
7. El chef teclea un tiempo estimado para completar la orden.
8. El sistema transmite el tiempo estimado a la palmtop del mesero.

Para continuar con las modificaciones que hemos hecho, posiblemente queramos cambiar el paso 5 a “El mensaje llega a la interfaz del cocinero en la PC de la cocina”. También podríamos hacer una entrevista a un chef o dos para preguntarles cómo calculan el tiempo estimado del paso 7. Quizá pudiéramos desarrollar un paquete de software que ayudara con ello.

La figura 20.3 hace los honores a este caso de uso.

FIGURA 20.3

El diagrama de secuencias de “Sondear el progreso de una orden”.



Implicaciones

Al ver todos los resultados obtenidos hasta ahora, los señores LaHudra, Nar y Goniff están eufóricos.

“Esto cambiará todo el entorno del negocio de los restaurantes”, dijo Nar.

“Sé que hemos logrado algo”, dijo LaHudra, “pero, ¿qué quieres decir con ‘cambiar todo el entorno del negocio de los restaurantes’?”

“Sí, ¿qué quisiste decir?”, preguntó Goniff.

“Bueno, si lo analizan”, respondió Nar, “todo el trabajo de un mesero cambiará, igual que el del chef. Los meseros no tendrán que correr de aquí para allá como ahora lo hacen. Siempre estarán al alcance de los clientes dado que siempre se encontrarán en sus áreas de servicio designadas. Irán a la cocina y al bar sólo cuando tengan que hacerlo. Mediante sus palmtops, se convertirán en supervisores del proceso de preparación de las órdenes y administradores de sus áreas. Serán más parecidos a salvavidas que a meseros tradicionales. De hecho, podrán sentarse ocasionalmente mientras se encuentren en sus áreas, dado que el ‘trabajo’ ya no involucrará andar a las carreras de forma despiadada.”

“¿Y qué con los chefs?”

“También serán más administrativos. Utilizarán sus equipos de cómputo para asignar órdenes a sus asistentes, y coordinarán lo que proceda en una cocina. Esto será estu-
pendo para grandes cocinas y restaurantes, ahora que lo que hacemos es mover infor-
mación y no a la gente.”

“Hum... Suena bien”, dijo LaHudra. “Aparentemente, cuanta más información nuevas,
moverás menos a la gente y lograrás más. No está mal.”

“No del todo”, dijo Goniff, ya maquinando la siguiente expansión del negocio.

Resumen

Luego del análisis del caso de uso, un equipo de desarrollo volvió su atención a los com-
ponentes del sistema que sugirieron los casos de uso. ¿Qué son? ¿Cómo interactúan?
Esta hora mostró cómo responder a estas preguntas dentro del contexto del desarrollo del
sistema RICO.

El objetivo de esto es dar información a los programadores que les facilite su trabajo.
Los resultados de este análisis deberán facilitar a los programadores la codificación de los
objetos del sistema y la forma en que se comunican entre sí.

Luego de modelar la cooperación entre componentes, el sistema ya está más cercano a
convertirse en realidad. Conforme modele la cooperación entre los componentes, podría
encontrarse con que será adecuado modificar los casos de uso.

Preguntas y respuestas

P Ha mostrado diversas modificaciones a los casos de uso. ¿En realidad así
ocurre en un proyecto?

R Indudablemente que sí. Ciertamente, los ejemplos podrían parecer algo amañados;
por ejemplo: en realidad posiblemente podríamos haber sabido de la base de datos
en el primer caso de uso antes de llegar hasta este punto. Sin embargo, la meta
es mostrarle que conforme se desarrolle nuestra noción, el modelo también se
desarrollará.

P ¿Por qué podría ser que los casos de uso originales no capturaran todos los
aspectos en primera instancia?

R Porque reflejan los resultados de las sesiones JAD con los usuarios del sistema, no
con los desarrolladores de sistemas. Verá que todas las adiciones y cambios estaban
relacionados con el sistema, no con el negocio. Una vez que acabe las sesiones con
los usuarios potenciales y que tenga la oportunidad de analizar los casos de uso, será
común que salgan a la luz modificaciones como esas.

Taller

Aquí será donde tendrá la oportunidad de practicar con el modelado de la cooperación entre los componentes de un sistema. Interactúe con el Apéndice A, “Respuestas a los cuestionarios”, para encontrar las respuestas. Tal vez necesite utilizar los componentes listados en esta hora para que le ayuden a continuar por encima y más allá de los ejercicios listados y hacer otros diagramas de secuencias y colaboraciones.

Cuestionario

1. ¿Cómo representaría a un objeto que se genera durante el curso de un diagrama de secuencias?
2. ¿Cómo se representa el tiempo en un diagrama de secuencias?
3. ¿Qué es la “línea de vida”?
4. En un diagrama de secuencias, ¿cómo muestra una “activación” y qué es lo que representa?

Ejercicios

1. Desarrolle un diagrama de colaboraciones equivalente al de secuencias para el caso de uso del Mesero “Tomar una orden”.
2. Cree un diagrama de secuencias para el caso de uso “Tomar una orden de bebida”.
3. Seleccione al menos un caso de uso del paquete Chef y desarrolle un diagrama de secuencias. Utilice la lista de componentes mencionados en esta hora. ¿Se necesitan algunos otros?
4. Válgase de su imaginación con ésta: Los casos de uso del paquete Encargado Del Guardarropa parecen ser muy sencillos. ¿Podría adornar a cada uno mediante la adición de uno o dos pasos? ¿Podrían ayudar algunos otros componentes adicionales? Dibuje un diagrama de secuencias para uno de estos casos de uso.
5. Continúe con su proyecto Biblioteca para listar las partes funcionales de su sistema y agregar la colaboración entre objetos. Genere los diagramas de secuencias para los casos de uso resultantes, una vez que haya hecho las modificaciones pertinentes a cada caso de uso.

HORA 21



Diseño del aspecto, sensación y distribución

Ahora nos centraremos en dos importantes aspectos del diseño de sistemas: la interfaz del usuario y la distribución del sistema.

En esta hora se tratarán los siguientes temas:

- Algunos principios generales del diseño de interfaces gráficas
- La sesión JAD de la GUI
- De los casos de uso a las interfaces de usuario
- Diagramas UML para el diseño de la GUI
- Esbozos de la distribución del sistema

Ya ha cumplido con éxito gran parte del análisis conducido por casos de uso. En esta hora, verá dos importantes aspectos del diseño de sistemas. Ambos pueden, a final de cuentas, orientarse a casos de uso, y son muy importantes para el producto final. Las GUI (interfaces gráficas de usuario) determinan qué tan práctico es un sistema. La distribución convierte a la arquitectura física planeada del sistema en una realidad.

Algunos principios generales en el diseño de las GUI

El diseño de las GUI, que conjuga al arte y la ciencia, dibuja, bajo la visión del artista gráfico, las conclusiones del investigador de factores humanos y las intuiciones del usuario potencial. Luego de mucha experiencia con interfaces WIMP (Ventanas, Iconos, Menús y Dispositivos apuntadores), han salido a la luz diversos principios generales. He aquí los principales:

1. Entienda lo que el usuario tiene que hacer. Por lo general, los diseñadores de interfaces realizan un *análisis de tareas* para comprender la naturaleza del trabajo del usuario. Nuestro análisis de casos de uso corresponde más o menos a esto.
2. Haga que el usuario sienta que tiene el control de la interacción. Siempre incluya la posibilidad de que el usuario cancele una acción luego que la haya iniciado.
3. Dé al usuario diversas formas de realizar cada acción relacionada con la interfaz (como el cierre de una ventana o archivo) y disculpe con elegancia los errores del usuario.
4. Dadas nuestras influencias culturales, nuestros ojos se dirigen a la esquina superior izquierda de la pantalla. Coloque la información de mayor prioridad allí.
5. Aproveche las relaciones de espacio. Los componentes de la pantalla que estén relacionados deberán aparecer uno junto de otro, quizá con un marco que los borde.
6. Destaque la legibilidad y la comprensión (¡palabras con las que tenemos que vivir!). Utilice la voz activa para comunicar ideas y conceptos.
7. Aunque tal vez tenga la capacidad de incluir miles de colores en la pantalla, restrinja los que vaya a utilizar. De hecho, haga una fuerte limitación de ellos. Demasiados colores distraerán al usuario de la tarea por realizar. Por cierto, se recomienda dar al usuario la opción de modificar los colores.
8. Si piensa en utilizar colores para indicar algo, recuerde que no siempre será fácil para el usuario asociar un color con un significado. A su vez, tenga en cuenta que ciertos usuarios (cerca del 10% de los adultos masculinos) sufren de daltonismo, y se les podría dificultar distinguir a un color de otro.
9. Como con el color, restrinja el uso de fuentes. Evite las fuentes cursivas y ornamentales. “Haettenschweiler” es un nombre de fuente que puede ser divertido pronunciar, pero que no fomenta una facilidad de uso.
10. Intente mantener, tanto como sea posible, los componentes (como los botones y cuadros de lista) del mismo tamaño. Si utiliza componentes de tamaños dispares, múltiples colores y fuentes, creará una terrible interfaz que los especialistas en el área conocen como un diseño de “pantalón de payaso”.
11. Alinee los componentes y campos de datos a la izquierda: alíneelos de acuerdo con sus bordes izquierdos. Esto reduce los movimientos oculares cuando el usuario tiene que revisar la pantalla.

12. Cuando el usuario tiene que leer y procesar información y luego hacer clic en un botón, coloque los botones en una columna a la derecha de la información, o en una fila debajo y a la derecha de la misma. Esto es conveniente debido a la tendencia natural (o cultural) de leer de izquierda a derecha. Si uno de los botones es el predeterminado, destáquelo y colóquelo como el primero en el conjunto.

Esta docena de principios no son los únicos, pero nos dan una idea de lo que está involucrado en el diseño de una GUI. El reto es comunicar la información adecuada en un contexto visual elemental, directo e intuitivo.



Para ver mayor información respecto al diseño de interfaces, en especial las que están relacionadas con Windows, visite <http://msdn.microsoft.com/UI>. En esa misma página encontrará información para el diseño de páginas Web o puede visitar www.useit.com. En ambos casos, la información se encuentra en inglés.

La figura 21.1 le muestra lo que ocurre cuando pone en acción algunos de estos principios y la figura 21.2 le muestra lo que sucede cuando no lo hace.

FIGURA 21.1

Aplicación de los principios para el diseño de GUI.

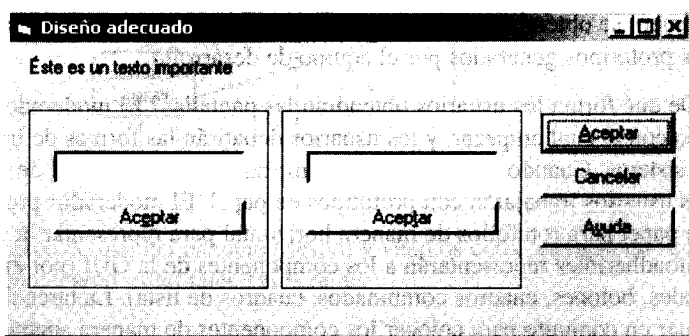
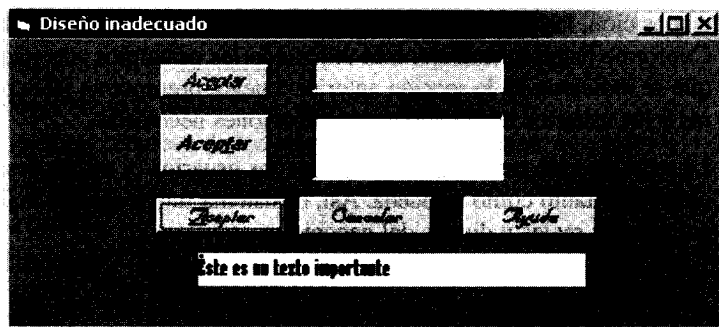


FIGURA 21.2

El resultado de no aplicar los principios para el diseño de GUI.



La sesión JAD para la GUI

Aunque la sesión JAD no tendrá una conexión directa con el UML, es buena idea hablar de la forma en que los usuarios potenciales determinan a la GUI. Nuevamente, se realizará una sesión para el Desarrollo conjunto de aplicaciones (JAD).

Para esta sesión reunirá a los usuarios potenciales del sistema. En el caso de RICO, reuniremos a los meseros, chefs, asistentes de los meseros, asistentes de los chefs, mozos de piso y a los encargados del guardarropa. Los integrantes del equipo de desarrollo que estarán presentes serán los programadores, analistas, modeladores y un moderador. El objetivo será comprender las necesidades de los usuarios e implementar una interfaz de acuerdo con sus ideas (una interfaz que integre sutilmente al sistema con los procesos del negocio). La antigua manera de desarrollar un sistema (por ejemplo, escribir un programa desde sus inicios, moldear el comportamiento del usuario para que puedan interactuar con él y modificar los procesos del negocio para que se adapten a él) ya es obsoleta.

Para mantener la eficiencia de la sesión, programará a los usuarios en grupos de acuerdo con el rol que desempeñen. Planeará la cantidad de tiempo que se llevará cada sesión de acuerdo con la cantidad de casos de usos en el paquete de cada rol. Esto es sólo un vago lineamiento, claro, ya que algunos casos de uso son más complejos que otros. Recuerde, a su vez, que podrían aparecer nuevos casos de uso al diseñar la GUI.

La participación de los usuarios en la sesión es un asunto que envuelve a dos partes. En la primera se obtendrán las pantallas de la interfaz del usuario. En la segunda se aprobarán los prototipos generados por el equipo de desarrollo.

¿De qué forma los usuarios obtendrán las pantallas? El moderador sugiere un caso de uso con el cual empezar, y los usuarios debatirán las formas de implementarlo mediante el sistema. Cuando estén listos para empezar a hablar al nivel de una pantalla específica, los usuarios trabajarán con prototipos en papel. El moderador proporciona una gran hoja de papel para rotafolios de manera horizontal para representar la pantalla. Las notas autoadheribles representarán a los componentes de la GUI (por ejemplo: menús contextuales, botones, cuadros combinados, cuadros de lista). La tarea de los usuarios será trabajar en conjunto para colocar los componentes de manera adecuada.

Cuando lleguen a un acuerdo en los componentes que deban estar en una pantalla y su disposición, los miembros del equipo de desarrollo generarán pantallas de prototipo. Conforme hagan su trabajo, irán utilizando los principios adecuados del diseño de GUIs que se indicaron en la sección anterior. Luego, presentarán tales pantallas en computadoras y los usuarios les harán las modificaciones necesarias.

La meta de todo esto, claro está, es que los usuarios (y no los desarrolladores) conduzcan el proceso tanto como se pueda. Así, el sistema funcionará de manera óptima en las actividades reales y diarias del negocio.

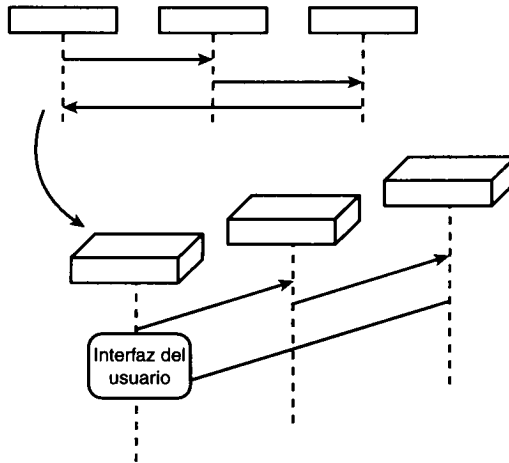
De los casos de uso a las interfaces de usuario

Los casos de uso describen la forma en que se utiliza el sistema. Por ello, la interfaz del usuario tiene que servir como medio para implementar los casos de uso.

Imagine un diagrama de secuencias de un caso de uso como una visión de un caso de uso. Si pudiéramos “girar” tal visión en tres dimensiones, para que la parte del extremo izquierdo del diagrama de secuencias se desprendiera de la página y quedara frente a nosotros, veríamos la interfaz que llevaría al usuario a la secuencia (vea la figura 21.3).

FIGURA 21.3

El supuesto “giro” del diagrama de secuencias nos pone de manifiesto la interfaz del usuario.



Examinemos los casos de uso del paquete Mesero y veamos cómo encajan en la interfaz de RICO. Nuevamente, aquí están tales casos de uso:

- Tomar una orden
- Transmitir la orden a la cocina
- Cambiar una orden
- Recibir una notificación de la cocina
- Sondear el progreso de la orden
- Notificar al chef del progreso de los clientes en sus alimentos
- Totalizar una cuenta
- Imprimir una cuenta
- Llamar a un asistente

- Llamar a un mozo de piso
- Tomar una orden de bebida
- Transmitir una orden de bebida al bar
- Obtener un acuse de recibo
- Recibir una notificación del bar

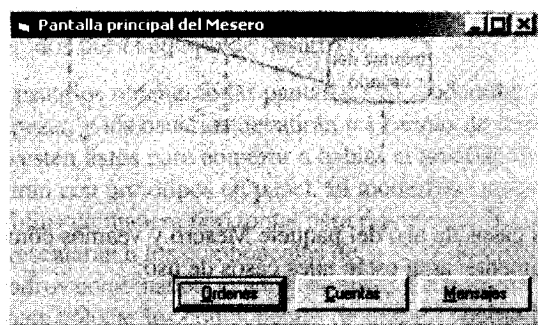
La interfaz del Mesero tiene que incluir a todos estos casos de uso.

Una forma de empezar es dividir al conjunto de casos de uso en grupos. Con tres de ellos será suficiente. Uno englobará a las órdenes (“Tomar una orden”, “Cambiar una orden”, “Sondear el progreso de la orden”, “Tomar una orden de bebida”). Otro grupo englobará a las cuentas (“Totalizar una cuenta”, “Imprimir una cuenta”). El tercer grupo englobará al envío y recepción de los mensajes (“Notificar al chef del progreso de los clientes en sus alimentos”, “Llamar a un asistente”, “Llamar a un mozo de piso”, “Transmitir una orden de bebida al bar”, “Obtener un acuse de recibo”, “Recibir una notificación del bar”).

Tal vez necesitemos empezar con una pantalla principal que llevaría al mesero por diversas pantallas hacia todos los demás grupos de casos de uso. Necesitamos tener la posibilidad de navegar de un grupo a cualquier otro. Dentro de un grupo, necesitamos navegar a cualquier caso de uso dentro del grupo. La figura 21.4 muestra una idea inicial de la pantalla principal. Esto tendrá que encontrarse en una palmtop, por lo que tal vez sea necesario ajustar su tamaño de alguna forma.

FIGURA 21.4

Primera idea de una pantalla principal del mesero.

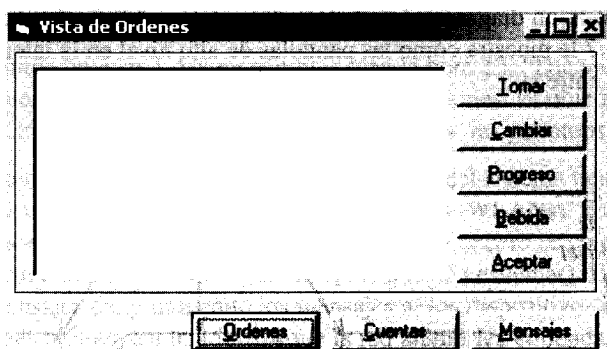


Nuestra sesión JAD podría llegar a convenir que la navegación dentro de un grupo se realizaría mediante botones localizados a la derecha de la pantalla, en tanto que la navegación entre grupos se realizaría con botones en la parte inferior de la pantalla. La figura 21.5 le muestra una idea básica de una de las interfaces del Mesero: la de los casos de uso relacionados con las órdenes.

Esta pantalla se abre en el modo Ordenes. El gran cuadro blanco se desplazará con una copia del menú con casillas de verificación que el mesero seleccionará para indicar las selecciones del cliente (cuando veamos la interfaz, tendremos un especial cuidado al utilizar la palabra “menú”). Al hacer clic en Aceptar se genera la orden y se envía a la PC de la cocina. Al hacer clic en un botón de la derecha se obtendrán sus características asociadas.

FIGURA 21.5

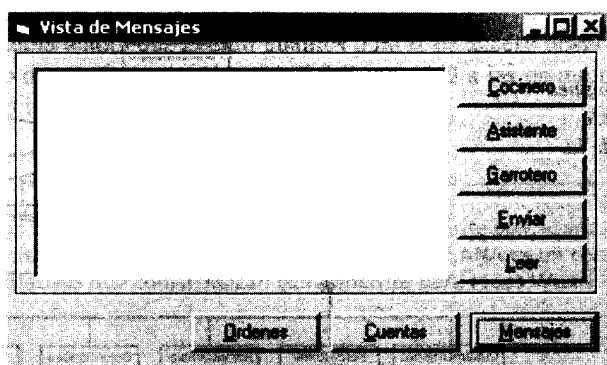
Pantalla para los casos de uso relacionados con las órdenes.



Cuando se haga clic en los botones de la fila inferior se obtendrá otro grupo de capacidades. Por ejemplo, el botón Mensaje traerá la pantalla de la figura 21.6. Por cierto la interfaz de usuario no debe ser sólo visual. Esta interfaz tiene que incorporar algún elemento audible para notificar a un mesero que le ha llegado un mensaje. Así, hará clic en el botón Leer para que se le presente una lista desplegable de mensajes.

FIGURA 21.6

Pantalla para los casos de uso relacionados con los mensajes.



Diagramas UML para el diseño de la GUI

El UML no hace recomendaciones específicas respecto a los diagramas para diseños GUI. No obstante, en una hora anterior indicamos una posibilidad: recuerde que en la hora 8, “Diagramas de estados”, presenté un ejemplo que trataba de los cambios de estado en una GUI. Aunque tal ejemplo profundizó en la funcionalidad de una GUI más de lo que deberíamos en este punto, sugiere que los diagramas de estados son útiles cuando se trata de interfaces de usuario.

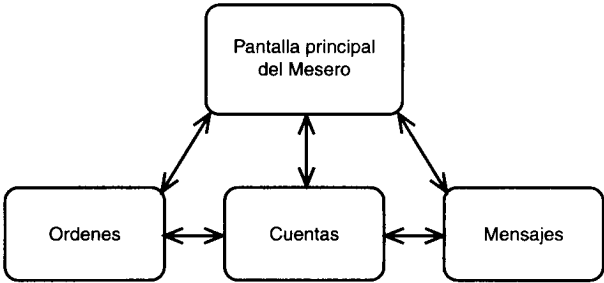


En la hora 24, “El futuro del UML”, presentaremos algunas ideas de cómo extender el UML para modelar las GUIs.

Debería usar un diagrama de estados para mostrar el flujo de una interfaz de usuario. La figura 21.7 le muestra cómo las pantallas de alto nivel en la interfaz del Mesero se conectan entre sí.

FIGURA 21.7

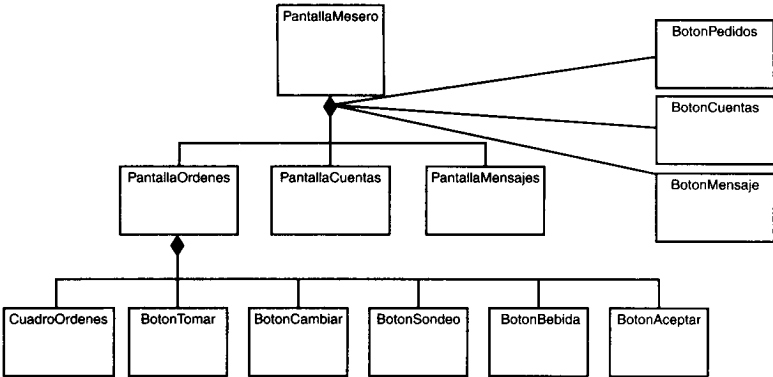
Un diagrama de estados para el flujo de pantalla de alto nivel en la interfaz del Mesero.



Como una pantalla en particular consta de varios componentes, un diagrama de clases para un objeto compuesto es adecuado para modelar una pantalla. La figura 21.8 muestra un diagrama para un objeto compuesto que corresponde con la pantalla de la figura 21.5.

FIGURA 21.8

Un diagrama compuesto que corresponde con la pantalla de la figura 21.5.



Esbozos de la distribución del sistema

Luego de que el segmento de análisis GRAPPLE ha producido el concepto general del sistema RICO, un ingeniero de sistemas empezará a pensar cómo deberá lucir la arquitectura física. Empezará a considerar las alternativas en topologías de red y la forma de implementarlas de manera inalámbrica, y comenzará a resolver qué componentes de software pertenecerán a determinados nodos en la red. Este segmento de diseño no tiene que esperar a que el análisis se haya completado. Sus acciones pueden proceder de forma paralela con las de los segmentos GRAPPLE, como con el diseño de la GUI.

La meta se centra en que el administrador del proyecto sondee todas las acciones en todos los segmentos.

La red

Al recordar los distintos tipos de LAN disponibles (vea la hora 13, “Diagramas de distribución”), el ingeniero del sistema cuenta con diversas opciones. El objetivo es elegir aquél que se integre mejor con la conectividad inalámbrica en las palmtops.

Para comprender algunas de las decisiones que el ingeniero del sistema tiene que tomar, vamos a profundizar un poco en las redes LAN inalámbricas (WLANs). Es frecuente el caso de que un transceptor de radio conocido como *punto de acceso* se coloque en un lugar fijo y se conecte a una LAN (por algún medio alámbrico estándar). El punto de acceso recibe los mensajes de, y transmite los mensajes a, los dispositivos inalámbricos, y cambia los mensajes recibidos a una forma que la red alámbrica entienda. Varios puntos de acceso aumentarán el rango de la WLAN y la cantidad de usuarios que podrán acceder a ella.

¿De qué manera los usuarios accederían a la WLAN? En RICO, harán la conexión mediante adaptadores LAN inalámbricos integrados con sus palmtops. No importa cuántos puntos de acceso se incorporen en la red, cada palmtop se asocia con sólo un punto de acceso y su área de cobertura, lo que se conoce como *microcelda* (que es similar a una celda que funciona en los teléfonos celulares).

El ingeniero del sistema tendrá que decidir cuántos puntos de acceso deberá tener el restaurante, qué tipo de adaptador inalámbrico LAN integrará a las palmtops, y el tipo y disposición de la red alámbrica.



Si algo de esto le ha interesado para su WLAN, visite www.wlana.com, el sitio Web de la Alianza inalámbrica LAN (WLANA). WLANA es un consorcio de corporativos que comercializan componentes para redes WLAN.

Vamos a suponer que el ingeniero del sistema decide utilizar una red thin ethernet para la LAN (vea la hora 13).

Los nodos y el diagrama de distribución

Ya enumeramos los nodos en nuestro sistema. Los meseros, asistentes de meseros y mozos de piso tendrán palmtops. La cocina, guardarropa y bar tendrán computadoras de escritorio. Cada computadora se conectará a una impresora. Además, cada área de servicio tendrá una computadora de escritorio conectada a una impresora para que pueda imprimir cuentas y obtenerlas sin caminar demasiado (un servidor de impresión, si se puede llamar así).

No obstante, en este punto veremos un ejemplo primordial del porqué es mejor empezar a pensar en las características de distribución desde el principio. Conforme se avanza, el ingeniero del sistema y el equipo de desarrollo tendrán que tomar una decisión de negocios: las palmtops requieren adaptadores LAN inalámbricos, que podrían ser muy costosos, y el software asociado podría estar fuera de las necesidades iniciales. Una posibilidad potencialmente menos costosa es la de cambiar los dispositivos móviles de palmtops a otras conocidas como Handheld que ejecuten Windows CE y utilicen una tarjeta PC para conectarse de manera inalámbrica a la LAN. Una característica importante de esta alternativa es que los programadores pueden utilizar herramientas de desarrollo ya conocidas para crear el software y llevarlo a las computadoras móviles.

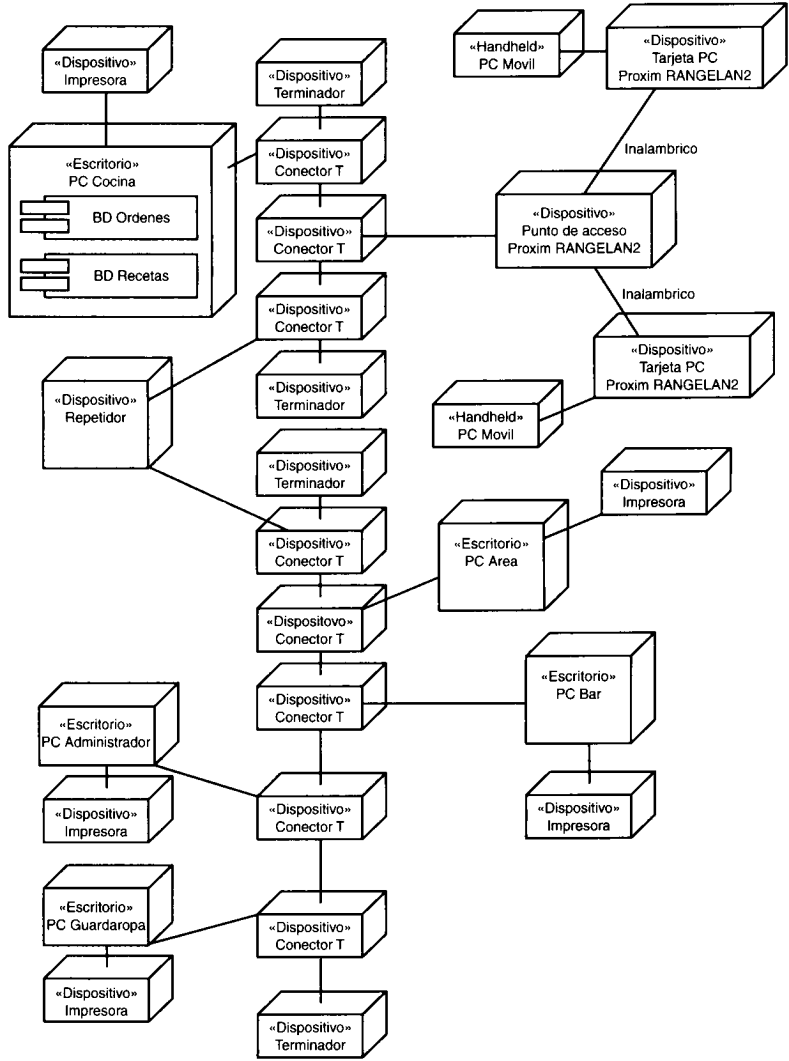
¿Por qué no utilizar una palmtop con Windows CE? Cuando escribí estas líneas, los dispositivos Palmtop con Windows CE no tenían ranuras para tarjetas PC, y la única tarjeta PC WLAN para Windows CE era la Proxim RANGELAN2. Conclusión: resuelva estos detalles en los procesos iniciales del proyecto.

En cualquier caso, asumamos que el equipo de desarrollo y los usuarios deciden utilizar la solución Windows CE/Proxim, con la idea de que el desarrollo será más eficiente y económico si los desarrolladores pueden aprovechar sus herramientas de desarrollo para crear el código necesario. Como comentario al margen, de acuerdo con la velocidad a la que avanzan las cosas, podría aparecer en cualquier momento un nuevo dispositivo Palmtop con Windows CE y una ranura para tarjetas PC.

Esta línea de razonamiento orienta al ingeniero del sistema hacia una solución Proxim para un punto de acceso a la LAN; al igual que la tarjeta PC, también se llama RANGELAN2.

Para ilustrar la distribución, el ingeniero del sistema genera el diagrama de distribución que se muestra en la figura 21.9.

FIGURA 21.9
Diagrama de distribución para RICO.



Siguientes pasos

El equipo de desarrollo ha ido de los casos de usos a las interfaces y a las WLANs. ¿Qué sigue?

Para empezar, los analistas pondrán en orden al modelo. Verán el diccionario de modelos y pondrán en orden cualquier ambigüedad. Se asegurarán de que toda la terminología sea utilizada de manera consistente en todos los diagramas y que los problemas con términos

como “menú” no propendan a confusiones. Cuando se completen todas las partes del análisis y diseño de GRAPPLE, el equipo compilará los resultados en un documento de diseño y enviará copias al cliente y a los programadores.

A los programadores o desarrolladores es a quienes corresponde convertir el diseño en código (la parte de la codificación va más allá de los alcances de este libro).

El código será probado, vuelto a escribir de acuerdo con los resultados de las pruebas, y vuelto a probar (proceso que continuará una y otra vez hasta que el código pase todas las pruebas). El análisis de los casos de uso conforma los fundamentos de las pruebas.

Los especialistas en la documentación empezarán a crear la información del sistema, y crearán también los materiales de entrenamiento. Un buen proyecto para la creación de la documentación debería proceder como uno para el desarrollo de un sistema, con una cuidadosa planeación, análisis y pruebas, y deberá comenzar en las primeras instancias del proceso de desarrollo.

Con un documento de diseño bien organizado e informativo y con un análisis y diseño sólidos, los siguientes pasos deberían proceder sin problemas durante todo el proceso de desarrollo.

La idea principal es enfocar los esfuerzos primordialmente en el análisis y diseño para que la implementación tenga muy pocos traspiés y que el proyecto dé por resultado un sistema que cumpla por completo con las necesidades del cliente.

...Y ahora, unas palabras de nuestros patrocinadores

Los señores LaHudra, Nar y Goniff no podrían estar más emocionados por la forma en que ha transcurrido el proyecto de desarrollo. El equipo de desarrollo los ha mantenido informados del proceso y les ha dado diseños basados en UML que les muestran el punto en que se encuentra el proyecto. Incluso están satisfechos por la forma en que el equipo solucionó el gran problema de distribución respecto a los dispositivos móviles.

Todo el esfuerzo ha despertado sus imaginaciones y los ha impulsado a buscar nuevas formas de utilizar la tecnología (tanto dentro como fuera del mundo restaurantero). Han caído en la cuenta que la mayoría de los procesos de negocios involucran la transmisión de la información. El grado en que la tecnología acelera tal transmisión, representa una enorme ventaja competitiva.

Mejorar el trabajo de la fuerza de ventas

Fuera del mundo restaurantero, los tres empresarios ven el potencial de volver a utilizar las ideas de la LAN inalámbrica para una fuerza de ventas móvil dentro de un enorme área de trabajo. El volverlas a emplear no debería ser difícil, ya que toda la información de los modelos está intacta.

Una aplicación de esta idea podría ser en las enormes tiendas departamentales o de autoservicio. Los vendedores de piso de dichas tiendas podrían beneficiarse de un dispositivo de mano que acceda a la información de los productos mediante una LAN inalámbrica. Un sistema como éste podría ayudar al vendedor a responder preguntas respecto al lugar donde se encuentra el producto en la tienda, si hay en existencia y cómo se podría utilizar.

Esto tiene algunas implicaciones intrigantes tanto para el vendedor como para el cliente. Los clientes estarían seguros que siempre obtendrán la información más reciente y concisa del vendedor. Un nuevo vendedor entrenado sobre cómo utilizar el sistema podría, con rapidez, empezar a trabajar aunque tuviera una mínima capacitación sobre el almacén.

LaHudra, Nar y Goniff pronto invadirán el mundo de las mejoras al hogar.

Expansiones en el mundo restaurantero

Esta idea de la fuerza de ventas móvil no es suficiente para LaHudra, Nar y Goniff. No quieren hacer nada que no incluya a la tecnología para revolucionar los negocios restauranteros. Creen que podrían crear restaurantes basados en RICO en las principales ciudades del mundo. Creen que la tecnología simplificará la sensación de comer fuera de casa y lo hará más conveniente para que todos salgan a hacerlo.

Goniff, como siempre en busca de nuevas formas de ganar dinero, tiene un buen rato pensado en esto (¡al menos desde que finalizó la hora 20!).

“Amigos,” les dijo a sus socios, “si construimos restaurantes en todas las ciudades principales, podríamos llevar la tecnología a un siguiente paso y transmitir la información por doquier.”

“¿Cómo?” preguntó Nar, que siempre ha sido algo torpe.

“Piénsenlo. Si nos internacionalizamos, podríamos entrar en la Web y...”

LaHudra interrumpe: “Un momento. Ya estamos en la Web. Hay muchos que visitan www.lahudranargoniff.com, ¿O no?”

“Déjame terminar, LaHudra. Podríamos utilizar la Web para que la gente ‘vaya’ a todos estos restaurantes. Utilizaríamos la Web para darles un emparedado gratuito.”

“¿Qué?” preguntaron Nar y LaHudra al unísono y con incredulidad.

“Vamos a ver. Orientamos una página de nuestro sitio Web a nuestra división de restaurantes. Alguien entra a la página, da su nombre y otra información, y se le pide que elija el emparedado que desee. Si nuestra base de datos muestra que el visitante no ha hecho esto con anterioridad, se le presentaría otra página donde podrá imprimir un cupón para un emparedado. Llevará el cupón al restaurante más cercano. Obtendrá el emparedado, lo comerá, le gustará y regresará como un cliente normal.”

“Sí, pero la Web puede verse en cualquier parte”, dijo Nar. “Supón que alguien no vive cerca de uno de nuestros restaurantes y, de todas maneras, desea el emparedado.”

“¡Aguarden! ¡Ya sé!”, dijo LaHudra. “Pueden utilizar su tarjeta de crédito para pagar un cargo nominal por envío en el sitio Web, y nuestro restaurante más cercano se lo enviará a su hogar en un frío y económico contenedor. Podrán colocar el emparedado en su horno de microondas y calentarlo. Así, podrán comer un producto de LaHudra, Nar y Goniff donde estén. Luego, cuando viajen a una ciudad donde haya uno de nuestros restaurantes, querrán comer allí.”

“Por cierto, ¿para qué sería el resto de la información que ellos capturen antes de imprimir el cupón?”, preguntó Nar.

“Te lo diré,” dijo Goniff. “Utilizaremos esta indagación para enviarles por correo electrónico información promocional de nuestros demás negocios, de acuerdo con su demografía (siempre y cuando indiquen que quieren recibirla).

“¿Dónde está el equipo de desarrollo? Tenemos mucho trabajo qué hacer.”

Resumen

Cuando su proyecto se encuentre en el segmento de diseño, habrá dos elementos por enfocar que serán la interfaz del usuario y la distribución del sistema. Ambos son finalmente conducidos por los casos de uso y de extrema importancia.

El diseño de la interfaz del usuario depende de una visión artística y de una investigación científica. Varios de los principios del diseño de la interfaz han salido a la luz luego de años de trabajo con interfaces WIMP. Esta hora le mostró algunos de ellos. Téngalos en cuenta cuando su equipo de desarrollo diseñe GUIs.

Los casos de uso conducen el diseño de la interfaz del usuario. El sistema tiene que permitir al usuario completar cada caso de uso, y la interfaz es la puerta de acceso hacia cada uno de ellos.

De forma simultánea con varios de los procesos del proyecto, el ingeniero de sistemas del equipo se orientará a la arquitectura física. La arquitectura está conducida por los casos de uso dado que el uso del sistema finalmente determina la naturaleza física y la disposición del mismo. El ingeniero de sistema otorga un diagrama de distribución UML que muestra los nodos, los componentes de software que haya en cada nodo y las conexiones entre nodos. Aunque los detalles de la distribución aparecen en las etapas avanzadas del proceso GRAPPLE, no hay razón para no empezar a pensar en ellos en etapas previas. Como se mostró en esta hora, pueden aparecer detalles fundamentales que necesitarán ser resueltos.

Luego que el sistema ha sido modelado, la información del modelado puede volver a utilizarse en diversos contextos. El modelo puede impulsar miles de ideas nuevas para negocios.

Preguntas y respuestas

- P Una vez que los usuarios hayan generado un prototipo en papel, ¿en realidad será necesario crear la pantalla y mostrárselas? Después de todo, ya generaron la pantalla en el papel y colocaron adecuadamente los componentes. ¿No podrían esperar y ver las pantallas en el sistema una vez que esté finalizado?**
- R** Es indudable que tendrá que mostrarles a los usuarios una pantalla verdadera (“verdadera” en el sentido de la computadora). Para empezar, es muy probable que los usuarios vean cosas en la pantalla que no vieron en el papel. Otra razón (relacionada con la primera) es que las dimensiones de las notas autoadheribles sólo se aproximan a las de los componentes en la pantalla. Colocar las notas autoadheribles podría distorsionar la relación de espacio entre los componentes de la pantalla. La pantalla podría lucir un tanto distinta al prototipo en papel. A su vez, las capturas de la pantalla se convertirán en partes muy valiosas para su documento de diseño.
- P Sé que esto no está directamente relacionado con el UML, pero uno de los principios de la GUI que mencionó fue dar al usuario diversos medios para realizar las acciones relacionadas con la interfaz. ¿Por qué esto es importante?**
- R** Lo es porque no podrá predecir todos los contextos en donde un usuario realizará una acción. En ocasiones el usuario utilizará una aplicación que utiliza mucho el teclado, y una combinación de teclazos será más adecuada que un clic con el ratón. En ocasiones el usuario utilizará mucho el ratón, y un clic será más adecuado. Dar ambos medios para realizar lo mismo hace que la interacción sea más fácil para el usuario.
- P Y hablando de preguntas que no se relacionan con el UML, quisiera saber el porqué del principio de la “voz activa” en la GUI.**
- R** Los estudios demuestran que las personas comprenden mejor la voz activa que la pasiva. A su vez, la voz activa requiere, por lo general, menos palabras y, con ello, ocupará menos del precioso espacio de la pantalla que la voz pasiva. Los usuarios (así como los editores y jefes de redacción) aprecian si sus instrucciones dicen: “Haga clic en el botón Siguiente para continuar” en lugar de “El botón Siguiente debería ser oprimido por usted para que continúe el proceso”.

Taller

Este taller verifica su conocimiento de las cuestiones relacionadas con el diseño del aspecto y sensación de un sistema, y para orientarse a la arquitectura física correspondiente. Diseñe bien sus respuestas y, luego, vaya al Apéndice A, “Respuestas a los cuestionarios” para verificarlas.

Cuestionario

1. ¿Qué es un análisis de tareas?
2. ¿Cuál análisis de los que ya hemos hecho es un vago equivalente de un análisis de tareas?
3. ¿Qué se entiende por un diseño de tipo “pantalón de payaso”?
4. Dé tres razones para restringir el uso del color en una GUI.

Ejercicios

1. Utilice un diagrama de estados UML para modelar la interfaz del usuario del chef.
2. Utilice papel y lápiz para diseñar al menos una de las pantallas de la interfaz del usuario del chef. Empiece por agrupar los casos de uso y, luego, cíñase a las convenciones de la sesión JAD. Si puede utilizar Microsoft Visual Basic, o alguna otra herramienta para el diseño visual de pantallas, intente utilizarla para completar este ejercicio.
3. Nuevamente, continúe con el análisis de las tareas que haya obtenido en su proyecto de la Biblioteca. No olvide hacer un análisis de la distribución.

HORA 22



Noción de los patrones de diseño

Ahora que ha comprendido los fundamentos del UML y ha visto cómo utilizarlo en el contexto de un proyecto de desarrollo, finalizaremos la parte II con una semblanza de la aplicación del UML a un área novedosa: los patrones de diseño.

En esta hora se tratarán los siguientes temas:

- Parametrización
- Patrones de diseño
- Cadena de responsabilidad
- Uso de nuestros propios patrones de diseño
- Ventajas de los patrones de diseño

En las 21 horas anteriores ha visto muy diversos temas. Desde los diagramas de clases hasta los de secuencias, desde los diagramas de estados hasta las sesiones JAD, la meta fue prepararlo para aplicar el UML en situaciones que ocurren frecuentemente en el mundo real.

Ahora, cambiaremos nuestro rumbo un poco. En esta hora exploraremos una de las aplicaciones del UML que, posiblemente, se harán más populares. Esta aplicación, la representación de patrones de diseño, captura la esencia de las soluciones que han funcionado una y otra vez en proyectos y situaciones reales.

Parametrización

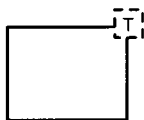
En la hora 2, “Orientación a objetos”, mencioné que una clase es una plantilla para la generación de objetos. Le dije que podía concebir a una clase como un molde de galletas generador de nuevos objetos. Un objeto, recordará, es una instancia de una clase.

Para ayudarle a recordar, veamos de nuevo el ejemplo de nuestra lavadora. La especificación de la clase lavadora (o, para utilizar una notación correcta, la clase Lavadora) incluía los atributos marca, modelo, numeroDeSerie y capacidad, y las operaciones agregarRopa(), agregarDetergente y quitarRopa(), con lo que tendríamos una forma de generar nuevos objetos provenientes de nuestra clase Lavadora. Cada vez que queramos crear un objeto, asignaremos valores a los atributos.

El propio UML le permite ir un paso más allá. Le da un mecanismo para crear clases de una forma similar a la creación de objetos. Podrá establecer una clase de manera que cuando asigne valores a un subconjunto de sus atributos habrá creado una clase en lugar de un objeto. A este tipo de clase se le conoce como clase *parametrizada*. Su representación en el UML aparece en la figura 22.1. El cuadro punteado del extremo superior derecho contiene los parámetros a los que asignará valores para generar la clase. A estos se les llaman *parámetros desvinculados*. Cuando les asigne valores, los vinculará con ellos. La T del cuadro punteado es un clasificador que indica que la clase es una plantilla para crear otras clases.

FIGURA 22.1

El icono UML para una clase parametrizada.



He aquí un ejemplo. Suponga que establece un SerVivo como clase parametrizada. Los parámetros desvinculados podrían ser géneros y especies, junto con los atributos estándar de nombre, estatura y peso, como se muestra en la figura 22.2.

Si vincula a género con “homo” y a especie con “sapiens”, creará una clase llamada “Humano”. El nombre de la clase se vincula con la T. La figura 22.3 le muestra una forma de representar tal vinculación. Este estilo en particular se conoce como *vinculación explícita* porque muestra de manera explícita la clase generada en una relación de dependencia con la clase parametrizada y le da a la clase generada su propio nombre.

FIGURA 22.2
SerVivo como una clase parametrizada.

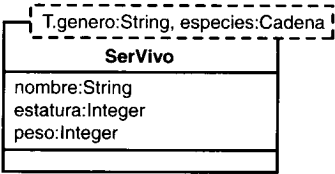
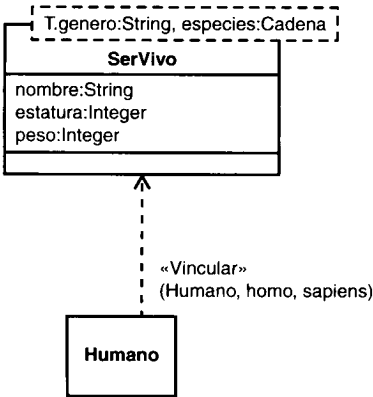
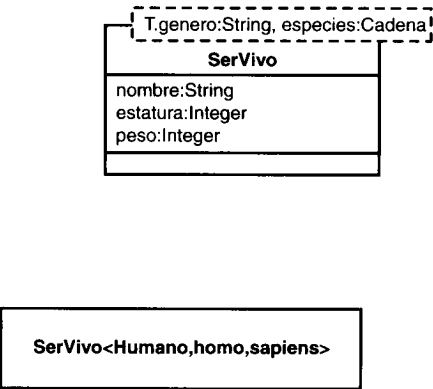


FIGURA 22.3
Vinculación explícita de la clase parametrizada SerVivo.



Hay otro tipo de vinculación que se conoce como *vinculación implícita*. En ella no se muestra la relación de dependencia, y las vinculaciones aparecen en una lista dentro de paréntesis angulares en el nombre de la clase generada. La figura 22.4 le muestra lo anterior.

FIGURA 22.4
Vinculación implícita de la clase parametrizada SerVivo.



En cualquier caso, podrá asignar valores a nombre, estatura y peso para generar objetos en la clase Humano.

Patrones de diseño

Es posible expandir la idea de la parametrización. Cualquier clasificador UML puede ser parametrizado. De hecho, un grupo de clasificadores que colaboren entre sí pueden ser parametrizados, lo que nos evitará entrar en situaciones desconcertantes.

Después de varias décadas de amplio uso, mismo que se incrementa, la orientación a objetos ha otorgado varias soluciones firmes a los problemas recurrentes. Tales soluciones se conocen como **patrones de diseño**, y han jugado un importante papel últimamente. Como los patrones de diseño han traspasado el mundo de la orientación a objetos, son fáciles de conceptualizar, diagramar y reutilizar. Al contar ahora con el UML, tenemos un lenguaje de modelado común para explicarlos y diseminarlos.

El primer libro que popularizó a los patrones de diseño fue “Design Patterns” (Addison-Wesley, 1995). Sus autores (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides) han sido reconocidos como el “Grupo de los cuatro”.

Un patrón de diseño es básicamente una solución (un diseño) que surge de la experimentación práctica con varios proyectos, y los equipos de desarrollo han encontrado que se puede aplicar en diversos contextos. Cada patrón de diseño describe a un conjunto de objetos y clases comunicadas. El conjunto se ajusta para resolver un problema de diseño en un contexto específico.

En su libro, el grupo de los cuatro catalogó y destacó a 23 patrones de diseño fundamentales. Dividieron esos patrones en tres categorías de acuerdo con cada uno de sus *propósitos*: (1) Patrones de *creación* que atañen al proceso de creación de objetos, (2) Patrones de *estructura* que se orientan a la composición de clases y objetos, y (3) Patrones de *comportamiento* que especifican la forma en que las clases u objetos interactúan y distribuyen la responsabilidad. Posteriormente dividieron sus patrones de diseño en términos si se aplican a las clases u objetos. A este criterio lo llamaron *ámbito*, y la mayoría de los ámbitos de los patrones se encuentra al nivel de los objetos.

Cada patrón de diseño tiene cuatro elementos: (1) un *nombre* que nos permite describir un problema de diseño en una palabra o frase, (2) un *problema* que define cuándo aplicar el patrón, (3) una *solución* que especifica los elementos que conforman al diseño y cómo colaboran, y (4) las *consecuencias* de aplicar el patrón.

Ahora veremos esas “situaciones desconcertantes” que indiqué con anterioridad: dentro de un modelo, podemos representar un patrón de diseño como una colaboración parametrizada en el UML. El patrón de diseño se expresa de una forma genérica, con nombres genéricos para los colaboradores. El asignar nombres específicos del dominio hace que el patrón se aplique a un modelo específico. La colaboración parametrizada le ayuda a visualizar los detalles específicos dentro del contexto del patrón.

Cadena de responsabilidad

Examinemos un patrón de diseño y verá lo que quiero decir.

La Cadena de responsabilidad es un patrón de comportamiento que se aplica a cierta cantidad de dominios. Este patrón se encarga de la relación entre un conjunto de objetos y una petición. Aplicará este patrón cuando más de un objeto pueda manejar una petición. El primer objeto en la cadena obtiene la petición y la resolverá o se la enviará al siguiente objeto de la cadena, hasta que uno de ellos pueda manejarla. El objeto que originalmente hizo la petición no sabe cuál de los objetos la manejará. El objeto que al final maneja la petición se conoce como un *receptor implícito*.

Los restaurantes están establecidos de esta forma, al igual que los distribuidores de automóviles cuando financian la adquisición de automóviles. Por lo general, en un restaurante un cliente no envía una petición directamente a un chef ni sabe qué chef preparará su platillo. En vez de ello, el cliente le da una orden a un mesero, el mesero se la lleva al chef, quien podrá cumplir con ella o dársela a uno de sus asistentes (de cualquier forma, así ocurre con los supuestos restaurantes de LaHudra, Nar y Goniff). Con un distribuidor de automóviles, el agente de ventas pasa el pedido de préstamo a diversas instituciones financieras hasta que alguna decide otorgar el préstamo.

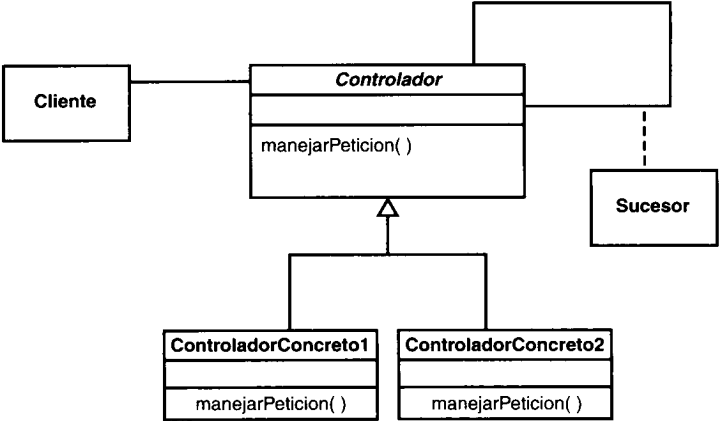
Las ligas de deportes profesionales implementan este patrón cuando un equipo pone transferible a un jugador. El equipo con la peor marca tendrá una oportunidad de solicitar al jugador. Si decide no hacerlo, el equipo con la siguiente peor marca tendrá la oportunidad, y así sucesivamente. El equipo que puso transferible al jugador no necesariamente sabe qué otro equipo se quedará con él (vea el Ejercicio 1).

Ahora que ha visto el patrón de diseño Cadena de responsabilidad en algunos contextos, ya está preparado para comprenderlo de manera abstracta. Los participantes en este patrón serán un Cliente, un Controlador abstracto y Controladores concretos que provengan del Controlador abstracto. El Cliente inicia una petición; si un Controlador (concreto) puede ocuparse de la petición, así lo hará; si no, pasará la petición al siguiente Controlador abstracto. La figura 22.5 le muestra la forma en que luce esta estructura.

La idea de este patrón es liberar a un objeto de que tenga que saber qué otro objeto realizará su petición. Le da una flexibilidad adicional cuando asigne responsabilidades a los objetos. La desventaja es que el patrón no garantiza que algún objeto maneje la petición. Por ejemplo: ningún equipo de fútbol contrataría a un jugador que haya sido puesto como transferible.

FIGURA 22.5

La estructura del patrón de diseño Cadena de responsabilidad.



Observe la asociación reflexiva en la clase **Controlador** abstracto. El grupo de los cuatro intentó mostrar que usted tendrá la opción de que el **Controlador** implemente un vínculo sucesor (en algunos contextos, los objetos saben cómo encontrar a sus sucesores). He decidido representar tal implementación con una clase asociación como en la figura 22.5, para permitir que más adelante se puedan agregar atributos al sucesor.

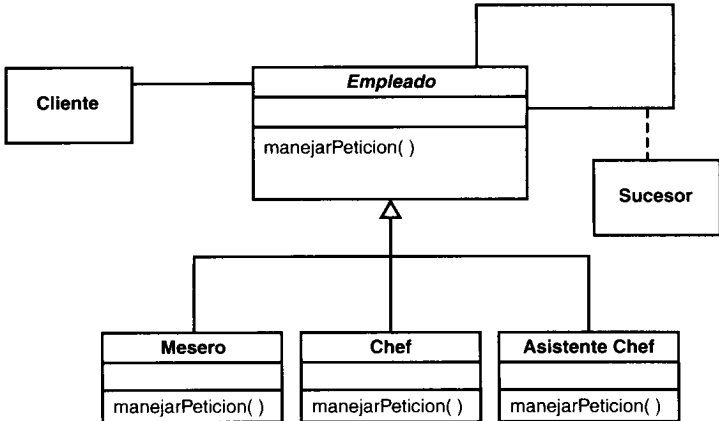
Cadena de responsabilidad: dominio Restaurante

En el dominio **Restaurante**, el **Controlador** abstracto es la clase **Empleado**, y los **Controladores** concretos son el mesero, el chef y el asistente. El **Cliente** es, en sí, el propio cliente, quien podría iniciar una petición, como hacer una orden, y no saber quién la llevará a cabo.

Al sustituir los nombres específicos del dominio de la figura 22.5 nos deja la figura 22.6.

FIGURA 22.6

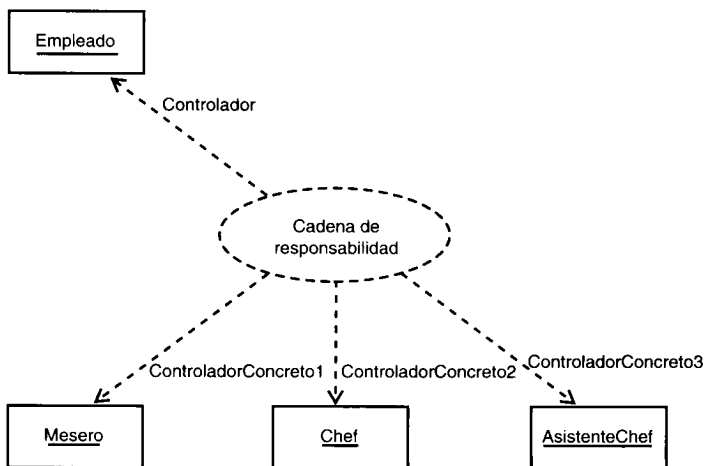
El patrón de diseño Cadena de responsabilidad en el dominio del restaurante.



La figura 22.6, aunque es un útil diagrama, no nos muestra la forma en que los nombres específicos del dominio encajan en el patrón. Para mostrar el contexto, utilizaremos una colaboración parametrizada, como en la figura 22.7.

FIGURA 22.7

Una colaboración parametrizada para representar la Cadena de responsabilidad en un restaurante.



En la figura 22.7, el óvalo punteado representa la colaboración que hay en el patrón de diseño, de allí el nombre dentro del óvalo. Los cuadros que lo rodean representan a los colaboradores. Las dependencias muestran que la colaboración depende de los colaboradores. La etiqueta en una dependencia indica el rol que juega el colaborador dependiente dentro del patrón. La colaboración se ha parametrizado con la adición de los nombres de clases específicas del dominio.

Cadena de responsabilidad: Modelos de eventos de los exploradores Web

Cuando se desarrollan páginas Web interactivas, un diseñador debe tomar en cuenta el modelo de eventos del explorador que las abrirá. En Internet Explorer, puede escribir código de JavaScript o VBScript para reaccionar a un evento, como el clic en un botón. Este código, conocido como “controlador de evento”, establece los cambios (en caso de haberlos) que habrá cuando se haga clic en un botón.

En un documento HTML, podrá dividir una página en áreas conocidas como DIV, y subdividir a un DIV en formularios. Puede colocar un botón dentro de un formulario. ¿Le suena a objeto compuesto? ¡Claro, porque eso es! Cada elemento es un componente del documento, y ciertos componentes son parte de otros. La Gamma lista a los objetos compuestos como uno de sus patrones de diseño, y se usa con frecuencia junto con el patrón de Cadena de responsabilidad. La relación entre componente y objeto compuesto implementa los vínculos de los sucesores. Cuando le mostré el diagrama de clases de la Cadena de responsabilidad, le mencioné a modo de aclaración que en ciertos contextos los objetos saben cómo localizar a sus propios sucesores. Éste es uno de esos contextos.

Cuando se coloca el botón en un formulario dentro de un DIV cuyo documento se abre en Internet Explorer, el evento de hacer clic en él iniciará con el botón en sí, pasará al formulario, luego al DIV y, finalmente, al documento que lo contiene. Cada uno de estos elementos puede tener su propio controlador del evento hacer clic en el botón, el cual reaccionará ante tal evento.

Si una secuencia de comandos que se encuentre en el documento establece de forma dinámica cuál de los controladores de eventos del elemento se ejecuta, la secuencia de comandos será una instancia del patrón de diseño de la Cadena de responsabilidad. La figura 22.8 le muestra el diagrama de clases, y la 22.9 le muestra la colaboración parametrizada para este patrón de diseño aplicada al modelo de eventos del Internet Explorer.

FIGURA 22.8
Diagrama de clases de la Cadena de responsabilidad en una página Web que se abre en Internet Explorer.

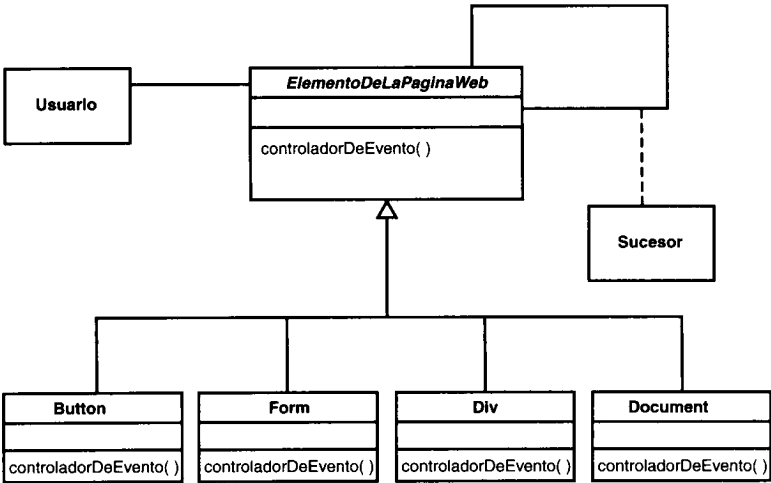
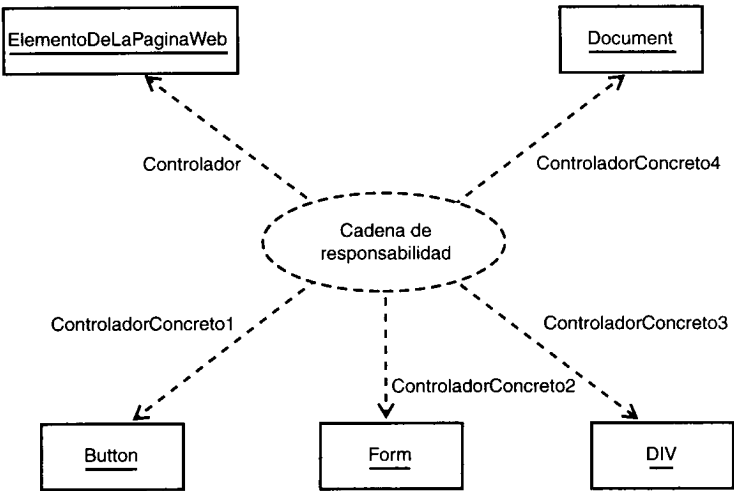


FIGURA 22.9
La Colaboración parametrizada para la Cadena de responsabilidad en una página Web que se abre en Internet Explorer.



Netscape Navigator también cuenta con un modelo de eventos. Su modelo es, exactamente, lo opuesto al de Internet Explorer. En Navigator, el elemento de mayor nivel (el documento) primero obtiene el evento y lo pasa hasta que lo lleva al que lo originó. ¿Cómo cambiaría al diagrama de clases de la figura 22.8 para modelar al modelo de eventos de Navigator? (Vea el Ejercicio 2.)



Internet Explorer llama a su modelo de eventos *burbujeo de eventos*. Navigator lo llama *captura de eventos*.

Nuestros propios patrones de diseño

Aunque el Grupo de los cuatro saltaron a la fama por su catálogo de patrones de diseño, no establecieron que sus patrones fueran los únicos posibles. Al contrario, intentaron alentar el descubrimiento y uso general y propio de los patrones.

Para darle una idea de la forma en que surgen tales patrones, recuerde lo que hicimos en la hora 11, “Diagramas de actividades”. Durante esa hora, vio un ejemplo de cómo tratar con los números de Fibonacci y un ejercicio referente a los números triangulares.

¿Cuáles fueron las características en común? Cada uno empezó con un valor o conjunto de valores iniciales, siguió con una regla para acumular números y finalizó con el *enésimo* número de la serie.

Llamemos a este patrón “Calculadora de series”. Aunque podríamos implementarlo en un objeto, hagámoslo en un conjunto de objetos colaboradores para ilustrar algunos conceptos de los patrones de diseño.

La Calculadora de series cuenta con tres participantes: ValorInicial (que puede contener uno o más valores), ReglaDeAcumulacion y ValorFinal. La figura 22.10 muestra el diagrama de clases de este patrón. El valor inicial está en un atributo llamado *primero*. Si es necesario un segundo valor inicial, como en el caso de los números de Fibonacci, se especifica en un atributo llamado *segundo*. En ocasiones, como en el caso de los factoriales, el patrón necesitará un valor para el término *cero*. El algoritmo para la ReglaDeAcumulacion se implementa en la operación *acumular()*. La cantidad de términos por calcular está en el *enésimo* atributo en ReglaDeAcumulacion.

En la colaboración, ReglaDeAcumulacion obtiene el (los) valor(es) inicial(es) de ValorInicial, aplica la regla la cantidad de veces indicada y envía el resultado a ValorFinal para que sea mostrado. La figura 22.11 muestra la interacción.

FIGURA 22.10

La estructura de clases para nuestro patrón de diseño Calculadora de series.

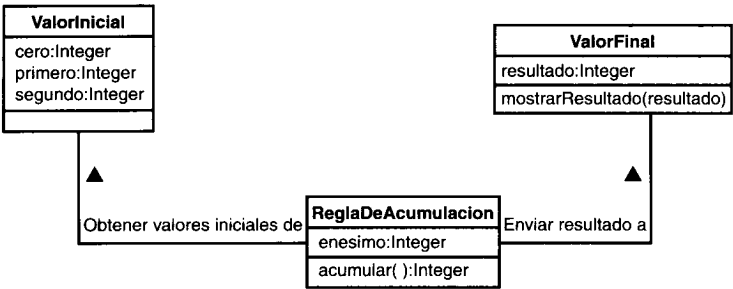
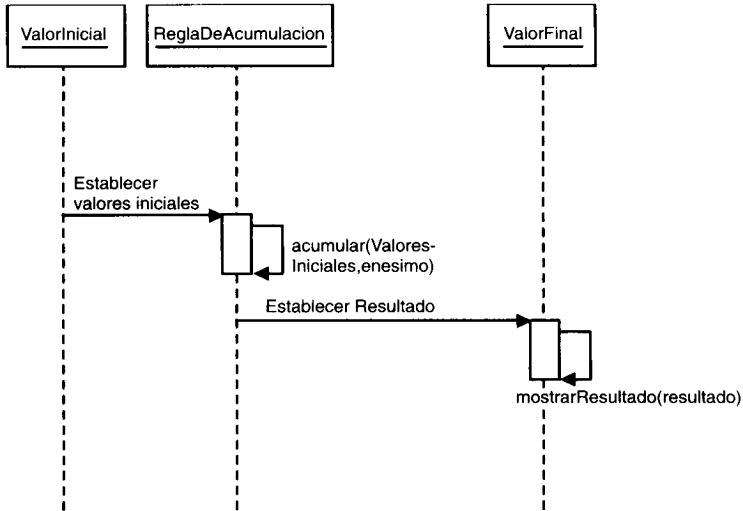


FIGURA 22.11

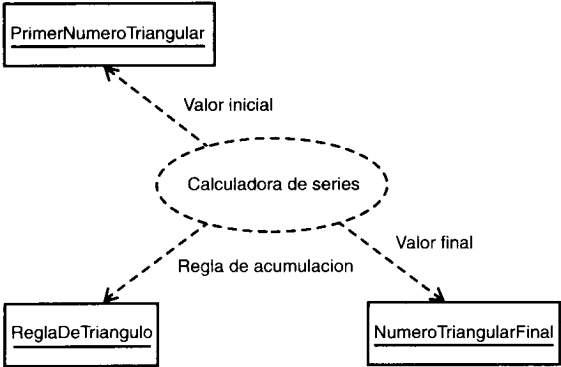
La interacción dentro del patrón de diseño Calculadora de series.



Para aplicar este patrón de diseño a la serie de números triangulares, adoptaremos algunos nombres de la numeración triangular para las clases y mostraremos la colaboración parametrizada de la figura 22.12.

FIGURA 22.12

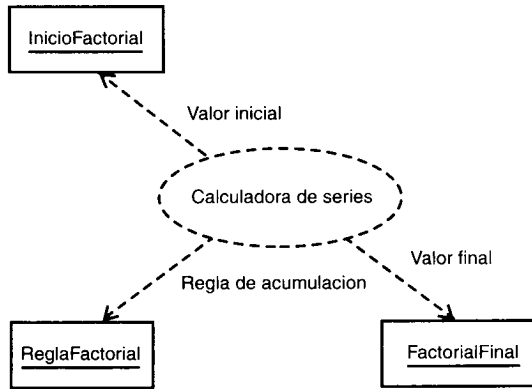
La colaboración parametrizada para la Calculadora de Números triangulares.



Como buena medida, mostraremos la colaboración parametrizada para una calculadora de factorial en la figura 22.13.

FIGURA 22.13

La colaboración parametrizada para una calculadora de Factorial.



Ventajas de los patrones de diseño

Los patrones de diseño son útiles en diversas maneras. Para empezar, promueven la reutilización. Si ha expresado un diseño bien fundamentado como patrón, lo habrá hecho sencillo para usted y otros que con él trabajen nuevamente. A su vez, le dan una forma clara y concisa de hablar y pensar respecto a un grupo de clases u objetos que funcionen en conjunto para resolver un problema. Esto aumenta la posibilidad de que utilice al patrón como un componente de un diseño. Finalmente, si utiliza patrones en su diseño, posiblemente se le facilitará documentar el sistema que haya generado.

Resumen

Una clase parametrizada tiene parámetros desvinculados. El vincular a tales parámetros dará como resultado la creación de una clase. Cualquier clasificador UML podrá estar parametrizado. Una colaboración parametrizada sirve como la representación de un patrón de diseño: una solución que es útil en diversos dominios.

Un patrón de diseño, la “Cadena de responsabilidad”, se ocupa de que los objetos pasen una petición entre ellos hasta que uno pueda manejarla. Este patrón proviene del libro más conocido en patrones de diseño, escrito por un grupo de autores conocidos como “El grupo de los cuatro”.

Nuestros propios patrones de diseño surgen del trabajo que hicimos en la hora 11 en los diagramas de actividades. Podemos crear un patrón de diseño para una calculadora que determine el enésimo valor de una serie aritmética. Los participantes en este patrón son ValorInicial, ReglaDeAcumulacion y ValorFinal.

Los patrones de diseño ofrecen varias ventajas. Permiten que los diseñadores vuelvan a utilizar con facilidad las soluciones ya probadas, incorporar componentes sólidos en los diseños y documentar claramente los sistemas que generen.

Preguntas y respuestas

P ¿Qué tan difícil es “descubrir” los patrones de diseño?

R No es cuestión de dificultad, sino de experiencia. Conforme avance en su carrera de analista y diseñador, verá que hay ciertas cosas que se repiten una y otra vez y después de un tiempo tratará de aprovechar esas circunstancias regulares. Los estudios muestran que los expertos en un dominio en particular piensan en términos de patrones y tratan de aplicarlos siempre que pueden.

P ¿Los patrones sólo son útiles en el diseño?

R No. Los patrones pueden surgir en cualquier momento del proceso de desarrollo o en cualquier campo de acción. El Grupo de los cuatro se inspiró en el trabajo de un arquitecto que pensó en recurrir a patrones para el diseño de los edificios.

Taller

Las preguntas y ejercicios en este taller le llevarán a pensar respecto a algunas características avanzadas del UML. Vaya al Apéndice A, “Respuestas a los cuestionarios”, para encontrar las respuestas.

Cuestionario

1. ¿Cómo representaría a una clase parametrizada?
2. ¿Qué es “vincular” y cuáles son los dos tipos de vinculación?
3. ¿Qué es un “patrón de diseño”?
4. ¿Qué es el patrón de diseño “Cadena de responsabilidad”?

Ejercicios

1. Aplique el diagrama de clases del patrón diseño Cadena de responsabilidad al proceso de transferencia que se realiza en las ligas deportivas profesionales.
2. Modifique el diagrama de clases de la figura 22.8 de modo que deje entrever el modelo de eventos de Netscape Navigator. Como ya lo indiqué anteriormente, un evento en Navigator se inicia al nivel del documento y pasa por los distintos elementos hasta que llega al que lo originó. El elemento originador podría localizarse a varios niveles por debajo del documento HTML.



PARTE III

Visión del futuro

Hora

- 23 Modelado de sistemas incrustados
- 24 El futuro del UML

HORA 23



Modelado de sistemas incrustados

Esta vez verá lo referente a sistemas de cómputo que no se basan en sistemas de escritorio, laptops o en palmtops. En vez de ello, estarán incrustados dentro de aparatos como aviones, trenes y automóviles.

En esta hora se tratarán los siguientes temas:

- ¿Qué son los sistemas incrustados?
- Conceptos de los sistemas incrustados
- Modelado de un sistema incrustado en el UML

LaHudra y sus intrépidos socios, Nar y Goniff, han tenido buenas ganancias de su División de Restaurantes LNG. El servicio es tan bueno y las comidas tan deliciosas que la gente llega por miles para probarlas dentro de una atmósfera de eficiencia y amistad.

Hay dos problemas que deterioran su otrora buena fortuna. Conforme leen los informes mensuales, la terrible tendencia ha continuado. “Miren esto”, dijo Nar dándoles los escritos a Goniff y LaHudra. “Estamos haciendo mucho dinero, pero deberíamos hacer más. Parece que los meseros están rompiendo más losa de la aceptable.”

“Sí, ya lo había advertido”, dijo Goniff. “Cada vez que rompen un plato lleno de comida, el cocinero tiene que preparar otra vez el plato. Y tenemos que pagar el nuevo plato.”

“¿Realmente importa si algunos de nuestros meseros tienen las manos torpes?”, preguntó LaHudra.

“Claro que sí”, respondió Goniff. “Un par de platos aquí, otros allá, pronto las pérdidas serán considerables. Pero hay algo de los meseros que me está molestando más.”

“¿Y qué es?”, preguntó Nar.

“Los reportes indican que se están enfermado demasiado. Es bueno eso de que tengamos toda esta tecnología en los restaurantes. Nos ayuda cuando tenemos que trabajar con limitaciones en el personal: por lo general los meseros pueden hacerse cargo de mayores secciones de lo que podrían normalmente.”

“Veamos qué ocurre”, dijo LaHudra.

La madre de la invención

Los tres socios entrevistaron a varios de los meseros que se habían enfermado con frecuencia en los dos últimos meses, e hicieron un asombroso descubrimiento: los platos rotos y los días de enfermedad estaban relacionados. Los meseros habían estado manejando y empuñando sus palmtops tanto que sus muñecas habían comenzado a debilitarse. Tal como una saliente puede hundir embarcaciones, una muñeca débil puede dejar caer los platos. Lo que es más, habían tenido tanto dolor en sus muñecas que no podían ir a trabajar.

“¿No podríamos ayudar a estas personas de alguna forma?”, preguntó desconsolado Nar.

“¿Y en el proceso ayudarnos a nosotros mismos?”, contra argumentó el oportunista Goniff.

“Tal vez haya alguna forma de que les ayudemos a mejorar sus fuerzas y sus muñecas”, dijo LaHudra.

“Bien, ¿qué haremos?”, preguntó Goniff, “¿comprar a cada uno un ejercitador de muñecas?”

“Podría ser peor”, dijo LaHudra, “pues no sé qué tan efectivos sean tales ejercitadores. Podría ser eterno el proceso de curación de nuestro personal”.

“Con todo, la idea es buena”, dijo Nar, “tal vez sólo necesitamos un ejercitador de muñecas mejor que el que se pudiera adquirir en algún establecimiento”.

“¿En verdad? ¿Cómo podríamos hacer un mejor ejercitador de muñecas?”, preguntó LaHudra.

No tuvo que esperar mucho la respuesta. Nar estaba en uno de sus papeles de querer patentar.

“Recuerdo que mucha gente cree que la mejor y más eficiente forma de ejercitarse es aquella que genera el mayor reto, cuando sus músculos están trabajando a su máxima potencia. Si podemos crear un ejercitador de muñecas que aumente su resistencia conforme los músculos del antebrazo trabajen más, apuesto que mejoraremos la fuerza de las muñecas de nuestros meseros en la mitad del tiempo que ocuparían con un ejercitador de muñecas común.”

“Y, ¿cómo haríamos eso?” se preguntaba el siempre pragmático LaHudra.

“Tal como revolucionamos al negocio de los restaurantes”, dijo Nar, “con tecnología”.

“A ver, espérame”, dijo LaHudra, “lo que hicimos en los restaurantes requirió el uso de computadoras. ¿Realmente quieres decirme que agregaremos una computadora a un ejercitador de muñecas?”

“¿Y por qué no?” dijo Nar.

“¡Cierto! ¿Por qué no?”, replicó Goniff. “Ya te entendí, Nar. Y cuando terminemos de generar este aparato, podríamos comercializarlo. Y ya tengo el nombre perfecto: ¿Qué tal ‘LNG TecnoApretón’?”

“Creo que me está gustando”, dijo, con cautela, LaHudra.

“Ya me gusta”, indicó Nar, con entusiasmo. “¿Dónde está ese equipo de desarrollo?”

Creación de TecnoApretón

El equipo de desarrollo de RICO se ha vuelto a reunir. Su nueva misión será la de generar TecnoApretón, una pulsera “inteligente” para la muñeca y el antebrazo que dé una resistencia variable durante los movimientos repetitivos de un ejercicio: cuanto más trabajen los músculos, mayor será la presión de TecnoApretón.

Durante la realización de la idea, el equipo hizo algunas investigaciones para ver cómo medir qué tanto trabaja un músculo. Aprendieron lo referente a señales eléctricas de las fibras activas de los músculos. Tales señales, llamadas EMG (siglas de señales Electromiográficas), son el fundamento de fascinantes dispositivos que permiten a los discapacitados manejar equipo electrónico.



Esto no será un tratado de ciencia-ficción. En los primeros años de la década de los noventa, el neurocientífico David Warner del Centro Médico de la Universidad Loma Linda colocó electrodos en el rostro de un muchacho y lo conectó a una computadora. El muchacho, completamente paralizado del cuello para abajo en un accidente automovilístico, pudo mover objetos en la pantalla de la computadora al tensar algunos de sus músculos faciales.

Para aprender más respecto a esta emocionante área, lea el artículo de Hugh S. Lusted y Benjamin Knapp, "Controlling Computers with Neural Signals", en la revista de octubre de 1996 de *Scientific American*.

El equipo concluyó que podrían capturar estos EMGs mediante pequeños y económicos "electrodos de superficie" colocados en el antebrazo, pasar los EMGs capturados por una computadora y, luego, utilizarlos como base para que la computadora ajuste la resistencia en el ejercitador de muñecas. Esto trae consigo la captura y análisis de datos en tiempo real, dado que los ajustes tienen que hacerse tan pronto como el músculo se contrae.

Una posibilidad del diseño sería colocar el electrodo de superficie en el antebrazo, conectarlo a una computadora de escritorio para que analice los EMGs y haga los ajustes necesarios al ejercitador de muñecas. La ventaja sería que posibilitaría la presentación de diversos datos en la pantalla, imprimir informes del progreso, y analizar las tendencias. No obstante, la desventaja sería que la persona quedaría atada a la computadora.

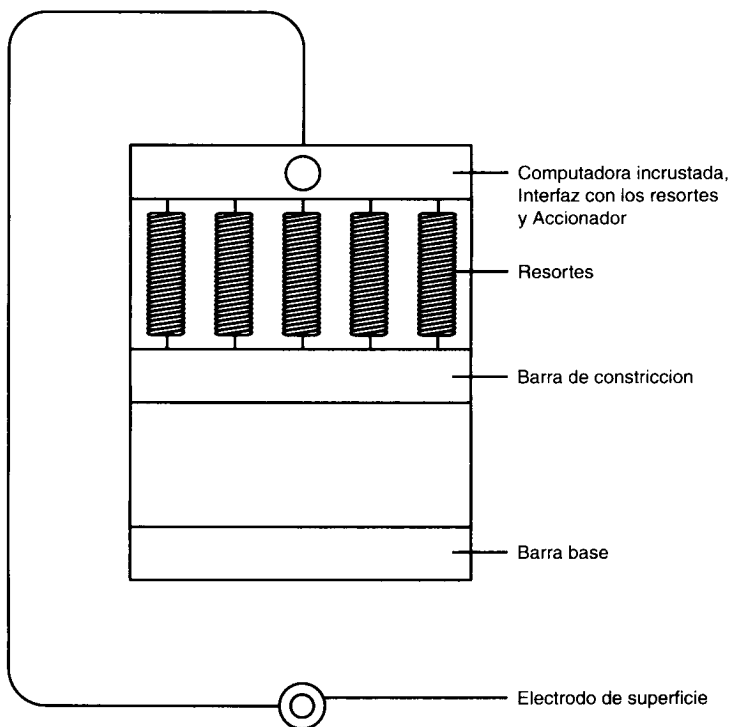
Otra posibilidad es la de incrustar un chip de computadora directamente en el ejercitador de muñecas para que la persona pueda moverse a donde desee mientras utilice TecnoApretón. La figura 23.1 muestra cómo luciría este diseño. En cada repetición del ejercicio, la persona aprieta la barra de constricción y la mueve hacia la barra base.

La ventaja del diseño incrustado sería que el usuario podría utilizar un dispositivo como éste en casi cualquier lugar (si la computadora tiene suficiente energía en las baterías). La desventaja, claro, es la pérdida de la información potencial que pudiera almacenar y mostrar una computadora de escritorio.

Las sesiones JAD dejaron entrever que todos serían más felices con el segundo diseño, y ello nos lleva hacia el mundo maravilloso de los sistemas incrustados.

FIGURA 23.1

*La versión en sistema
incrustado de
TecnoApretón.*



¿Qué es un sistema incrustado?

A estas alturas, ya sabe que las computadoras están por doquier. Lo que tal vez no sepa es qué tanto territorio abarca ese “doquier”. Las computadoras que ve a su alrededor son sólo la punta del témpano. Muchas de ellas se encuentran ocultas en lugares de difícil acceso: dentro de los electrodomésticos, automóviles, aviones, maquinaria de fábrica, dispositivos biomédicos y otros. Hay procesadores medianamente poderosos dentro de las impresoras.

Todas esas computadoras que no se ven a simple vista son ejemplos de sistemas incrustados. Siempre que tenga un dispositivo “inteligente”, tendrá un sistema incrustado.

Los sistemas incrustados no tienen teclados y monitores que interaccionen con nosotros, en vez de ello, cada uno es un chip que se encuentra dentro de un dispositivo (como un aparato electrodoméstico), y ese dispositivo no se parece para nada a una computadora. El sistema incrustado decidirá qué debe hacer el dispositivo.

Si utiliza un sistema de este tipo, no tendrá la sensación de trabajar con una computadora. En vez de ello, estará interactuando con un dispositivo. Le apuesto que nunca sabrá que hay un chip de computadora dentro. Cuando tuesta una rebanada de pan, no le preocupa que un chip de computadora esté distribuyendo el calor —sólo quiere su pan tostado.

Cuando termina de trabajar con su computadora, la apaga. Un sistema incrustado por lo general no se puede dar esos lujos. Una vez que se le coloca, un sistema tiene que trabajar por días o años (como en un marcapasos).

Si un procesador de textos o una hoja de cálculo tienen un error y su sistema se colapsa, simplemente vuelve a arrancarlo. Si el software en un sistema incrustado falla, los resultados pueden ser desastrosos.

Por lo tanto, un sistema incrustado no realiza el cómputo de la forma habitual. Se le coloca para ayudar a otro tipo de dispositivo a hacer su trabajo. El otro dispositivo es el que es manejado por el usuario y se integra al entorno.

Como podrá imaginar, programar un sistema incrustado no es para novatos. Requiere mucho conocimiento del dispositivo donde se encontrará el sistema: qué tipo de señales enviará, qué tipo de parámetros de cronometraje tendrá y otras cosas.

Conceptos de los sistemas incrustados

Veamos de cerca los sistemas incrustados y lo que comúnmente tienen que hacer. En las siguientes subsecciones examinaremos algunos de los conceptos más importantes de un sistema incrustado.

Tiempo

Si revisa el tema hasta donde vamos, verá que el *tiempo* se destaca en el mundo de los sistemas incrustados. De hecho, el tiempo es la base para clasificar a los sistemas incrustados como *tolerantes* o *estrictos*.

Un sistema tolerante hace su trabajo tan pronto como sea posible sin tener que cumplir con plazos específicos. Un sistema estricto también tiene que hacer su trabajo tan pronto como sea posible, pero deberá finalizar sus tareas de acuerdo con plazos rigurosos.

Subprocesos

En el mundo de los sistemas incrustados, un subproceso (que también se conoce como *tarea*) es un simple programa. Es una sección de una aplicación, y realiza cierto trabajo significativo dentro de ella. Intenta obtener toda la atención de la CPU. La *multitarea* es el proceso de programar el tiempo de la CPU para que trabaje con varios subprocesos y atienda a uno y otro.

Cada subproceso cuenta con un número que establece su prioridad dentro del programa de aplicación, y que se relaciona —por lo general— con uno de seis estados:

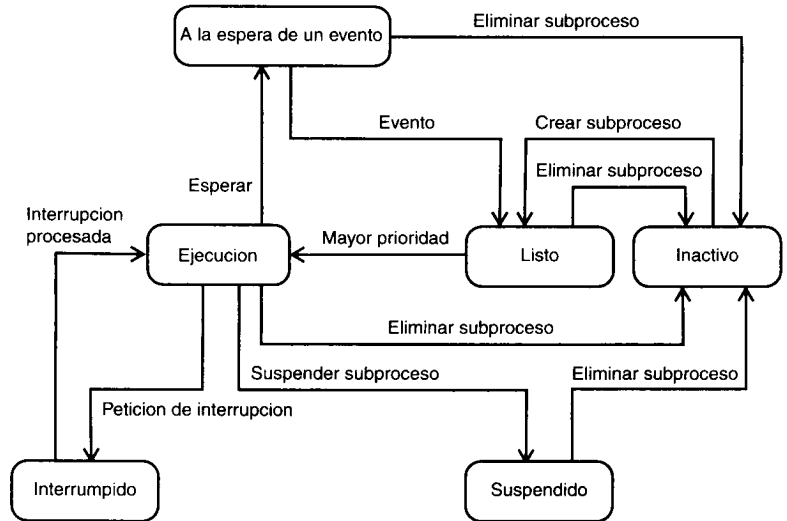
- *inactivo*: está en la memoria y no se ha hecho disponible al sistema operativo
- *listo*: puede ejecutarse, pero el subproceso que está en ejecución tiene una prioridad mayor

- *suspendido*: se ha interrumpido a sí mismo por cierto tiempo
- *a la espera de un evento*: algo debe ocurrir para que se ejecute
- *ejecución*: cuando está siendo atendido por la CPU
- *interrumpido*: porque la CPU se está ocupando de una *interrupción*

La figura 23.2 le muestra un diagrama de estados que representa a estos estados y las transiciones entre ellos. Observe la ausencia de un símbolo de inicio y otro de fin. Esto le indica que el subproceso fluye de un estado a otro en un ciclo infinito.

FIGURA 23.2

Estados de un subproceso en una aplicación de un sistema incrustado.



Por cierto, ¿qué es una “interrupción”? Continúe leyendo.

Interrupciones

Una *interrupción* es un elemento pequeño de mucha importancia en un sistema incrustado. Es un mecanismo basado en hardware que le indica a la CPU que ha acontecido un evento asincrónico. Un evento es asincrónico si sucede de forma impredecible (esto es, fuera de sincronía). Por ejemplo: en el TecnoApretón, las señales EMG llegan de forma asincrónica.

Cuando la CPU reconoce una interrupción, guarda lo que estaba haciendo e invoca a un ISR (Rutina para el servicio de interrupciones) que procesa al evento. Cuando el ISR finaliza su trabajo, la CPU continúa con lo que hacía cuando sucedió la interrupción.



Después de procesar una interrupción, el lugar a donde regresará la CPU se determina por el tipo de sistema operativo que en ella se ejecuta, como lo verá posteriormente.

Las interrupciones son importantes dado que permiten que una CPU se libere de lo que está haciendo y procese los eventos conforme suceden. Esto es tremendamente significativo para un sistema en tiempo real que tenga que responder a eventos del entorno de manera oportuna.

Ya que la puntualidad es tan importante, los sistemas incrustados tienen que ocuparse del curso del tiempo en una interrupción y su procesamiento, aunque tal lapso pudiera parecer infinitesimal. La CPU tiene que tomar algo de tiempo desde que se le notifica de la interrupción hasta que guarda lo que estaba haciendo (esto es, su *contexto*). A ello se le conoce como *latencia de la interrupción*. La *respuesta de la interrupción* es el tiempo entre la llegada de la petición de interrupción y cuando la CPU inicia el ISR. Al finalizar el ISR, la *recuperación de la interrupción* es el tiempo que le lleva a la CPU regresar a donde estaba —a su contexto— cuando ocurrió la interrupción.

Hay un tipo de interrupción especial: los *ciclos del reloj*. Como un “latido de corazón” del sistema, el ciclo del reloj sucede en intervalos regulares específicos de una aplicación (por lo general entre 10 y 200 microsegundos). Los ciclos del reloj determinan las restricciones de tiempo de un sistema incrustado. Por ejemplo: un subproceso en estado suspendido permanecerá así durante una cantidad específica de ciclos.

Sistema operativo

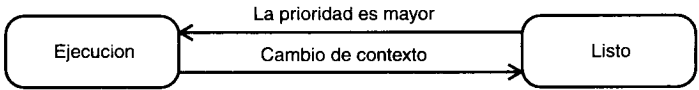
Un sistema operativo en tiempo real (RTOS) funciona como un policía de tránsito entre los subprocesos y las interrupciones; media la comunicación entre subprocesos y entre una interrupción y un subproceso. El núcleo o *kernel* es la parte del RTOS que administra el tiempo que utiliza la CPU en subprocesos individuales. El *programador* del núcleo determina cuál subproceso se ejecuta después. Como lo mencioné, cada subproceso tiene una prioridad asignada.

Los núcleos pueden ser *preferenciales* o *cooperativos* (no preferenciales), de acuerdo con la forma en que se encarguen de las interrupciones. En un núcleo cooperativo cuando se termina la ejecución de un ISR, la CPU regresa al subproceso en que se encontraba cuando llegó la petición de interrupción. Un núcleo cooperativo se dice que se involucra en la multitarea cooperativa. La figura 23.2 se aplica al núcleo cooperativo.

Por otro lado, en un núcleo preferencial, cuando finaliza un ISR, la CPU verifica la prioridad de los subprocesos que se encuentren en el estado de Listo. Si uno de los subprocesos tiene una mayor prioridad que la tarea interrumpida, la CPU ejecutará dicho subproceso en lugar de aquél en el que se encontraba cuando se recibió la petición de interrupción. Por ello, la tarea de mayor prioridad tiene preferencia sobre la tarea interrumpida. La figura 23.3 muestra la modificación a dos de los estados en la figura 23.2, para modelar al núcleo preferencial.

FIGURA 23.3

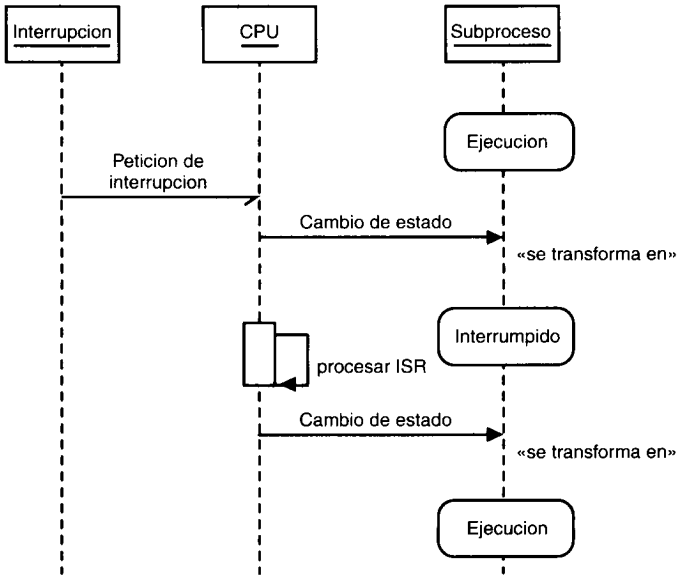
Modificación de transiciones entre dos estados en la figura 23.2 para reflejar lo que ocurre en un núcleo preferencial.



La Figura 23.4 modela el núcleo cooperativo como un diagrama de secuencias, y la figura 23.5 hace lo propio para el núcleo preferencial.

FIGURA 23.4

Diagrama de secuencias para el núcleo cooperativo.

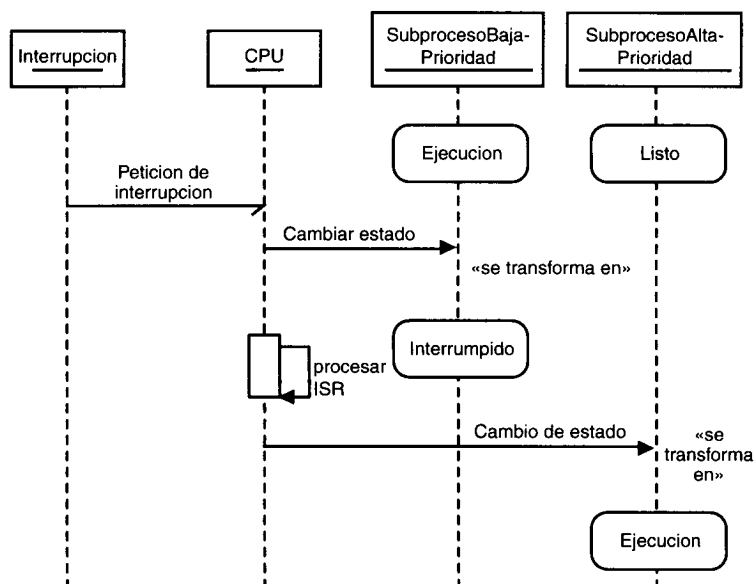


Microsoft empezó a reducir la brecha hacia el dominio de los sistemas incrustados. Está comercializando, de Windows CE, el sistema operativo para dispositivos de mano, al igual que para sistemas incrustados. Windows CE está generado de forma modular. Para ajustarse a una CPU en particular y a una aplicación incrustada, sólo requiere utilizar los módulos que necesite para que funcione.

La ventaja de utilizar Windows CE es que podrá utilizar el popular conjunto de herramientas de Microsoft junto con Visual C++ para generar un sistema incrustado.

FIGURA 23.5

Diagrama de secuencias para el núcleo preferencial.



Aunque hemos visto bastantes fundamentos, tenga en cuenta que sólo hemos visto cuestiones superficiales de los sistemas incrustados.

Modelado de TecnoApretón

De regreso a la tarea (¿subproceso?) que dejamos suspendido, empezaremos a crear un modelo para TecnoApretón. Aunque no es el caso de que todos los sistemas incrustados estén orientados a objetos, aun podemos utilizar tal orientación para modelar al sistema y su comunicación con el mundo exterior.

A partir de nuestro tema de los sistemas incrustados, es claro que tenemos que tomar en cuenta el cronometraje, los eventos, los cambios de estado y las secuencias.

Clases

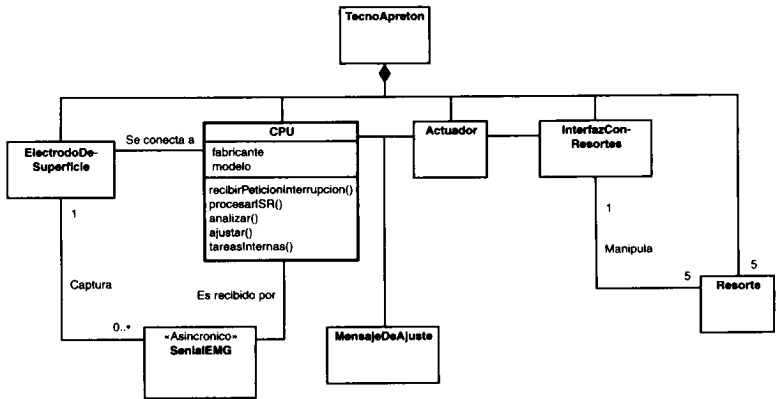
Como en el caso de cualquier otro tipo de sistema, empezaremos con las clases. Para comprender la estructura de clases, empezaremos con una descripción resumida de TecnoApretón y la forma en que funciona. Este resumen pudo haber sido el resultado de un análisis del dominio.

He aquí la descripción. TecnoApretón consta de un electrodo de superficie, una CPU, un accionador (para realizar los comandos de ajuste provenientes de la CPU), y un conjunto de cinco resortes. El accionador se conecta a los resortes mediante una interfaz mecánica. TecnoApretón recibe señales EMG asincrónicas de un electrodo de superficie y las pasa a la CPU. Cada EMG provoca una petición de interrupción, misma que la CPU tratará

con un ISR. El software en la CPU analiza las señales. Cuando el análisis está completo, la CPU envía una señal a un accionador para ajustar la tensión en los resortes. El actuador hace el ajuste mediante el manejo de la interfaz mecánica con los resortes.

La figura 23.6 muestra una estructura de clases que resume al párrafo anterior. Observe que el borde de la CPU está en negritas, lo que indica que es una clase activa. La idea es que la CPU siempre tendrá el control (en la recepción, análisis de señales y la administración de los ajustes). También realiza algunas tareas internas del sistema. Observe, a su vez, el uso de una clase de asociación para MensajeDeAjuste, que nos permite establecer parámetros al mensaje.

FIGURA 23.6
Estructura de clases de TecnoApretón.



SenialEMG y MensajeDeAjuste son importantes, así que nos enfocaremos en ellos. El sistema estará interesado en el momento en que llegue una señal y su potencia, por lo que horaDeLlegada y amplitud parecen ser atributos razonables para SenialEMG. A su vez, la SenialEMG tendrá indudablemente características complejas que están fuera del ámbito de este tema.

Para MensajeDeAjuste, tanto horaDeGeneracion y cantidadAjuste parecen ser atributos razonables.

La figura 23.7 muestra los atributos de estas clases.

FIGURA 23.7

Una mirada más
cercana a SerialEMG
y MensajeDeAjuste.

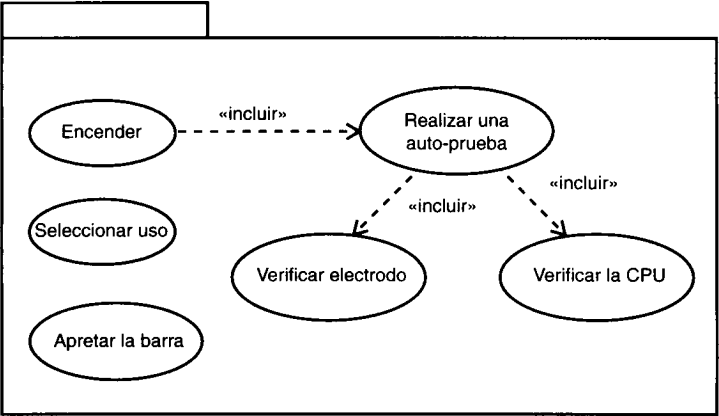
«Asincronico» SerialEMG	MensajeDeAjuste
horaDeLlegada amplitud caracteristicasDeSenial	horaDeGeneracion cantidadAjuste

Casos de uso

La sesión JAD que mencioné anteriormente (que dio por resultado la decisión de diseñar un sistema incrustado en lugar de una computadora de escritorio), también dio por resultado varios casos de uso, como se aprecia en la figura 23.8.

FIGURA 23.8

Los casos de uso de
TecnoApretón.



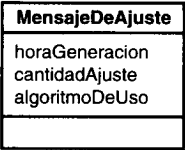
Estos casos de uso determinan las facultades por generar en el sistema. “Encender” incluye a “Realizar una auto-prueba” la cual, a su vez, incluye “Verificar el electrodo” y “Verificar la CPU”.

“Seleccionar uso” se refiere a diversas formas de ajustar el TecnoApretón, muchas de las cuales no se le ocurrieron al Sr. Nar cuando pensó en este dispositivo. Por ejemplo: los participantes de la JAD dijeron que les gustaría establecer repeticiones “negativas”: una resistencia limitada cuando aprieten las barras, y una máxima cuando dejen de apretar.

Esto significa que debemos agregar un atributo a MensajeDeAjuste para reflejar el uso del sistema. Podríamos llamarlo algoritmoDeUso y darle los valores posibles de aumentarTension y repNegativa. La figura 23.9 muestra la clase MensajeDeAjuste modificada.

FIGURA 23.9

La clase
MensajeDeAjuste
modificada.



Interacciones

Ahora prestemos atención a “Apretar la barra”, y asumamos que la persona ha seleccionado el modo originalmente concebido: incrementar la resistencia al incrementarse la actividad muscular. En esta parte del modelo, tenemos que asegurarnos de considerar restricciones de tiempo y cambios de estado. Asumamos que un intervalo de ciclos del reloj es de 20 microsegundos, y que el lapso desde que se recibe una señal hasta enviar un mensaje de ajuste no deberá ser mayor de 10 ciclos del reloj.

Una conjetura más: supongamos que el núcleo RTOS funciona de forma preferencial. Ello requiere de algunas pequeñas decisiones de modelado. Para empezar, y para reflejar la operación del núcleo, vamos a tratar las operaciones de analizar(), ajustar() y tareasInternas() de la CPU como subprocesos, y les asignaremos prioridades.

Para mostrarlas en nuestro modelo, tenemos que tratarlas como clases (algo que no solemos hacer con las operaciones). Este es un ejemplo de UML avanzado llamado *concreción*: tratar algo como si fuera una clase (u objeto) que no se suele tratar así. Cuando lo haga, enriquecerá a su modelo porque sus clases concretadas tendrán relaciones con otras clases, tendrán sus propios atributos y se convertirán en estructuras que podrá manipular y almacenar. En este caso, la concreción nos permite mostrar las prioridades de los subprocesos como atributos y utilizar los subprocesos en nuestros diagramas de interacción.

La figura 23.10 muestra la estructura de clases de los subprocesos de TecnoApretón.

FIGURA 23.10

Estructura de clases
para los subprocesos
de TecnoApretón.

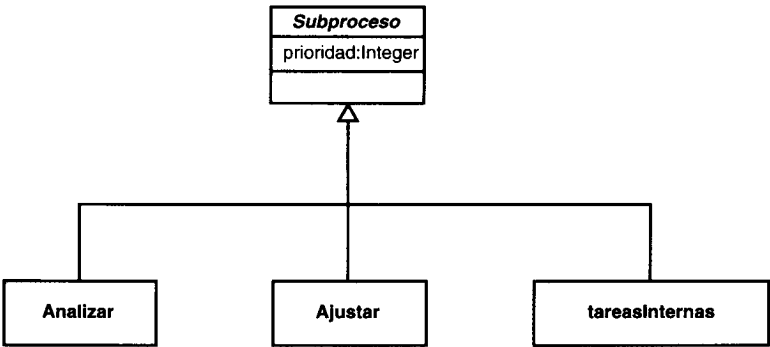
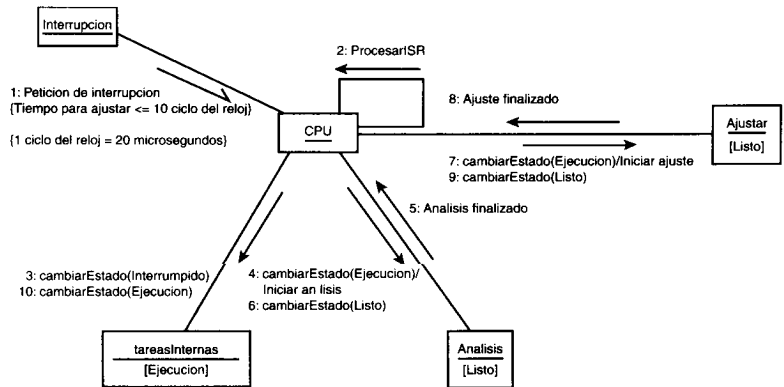


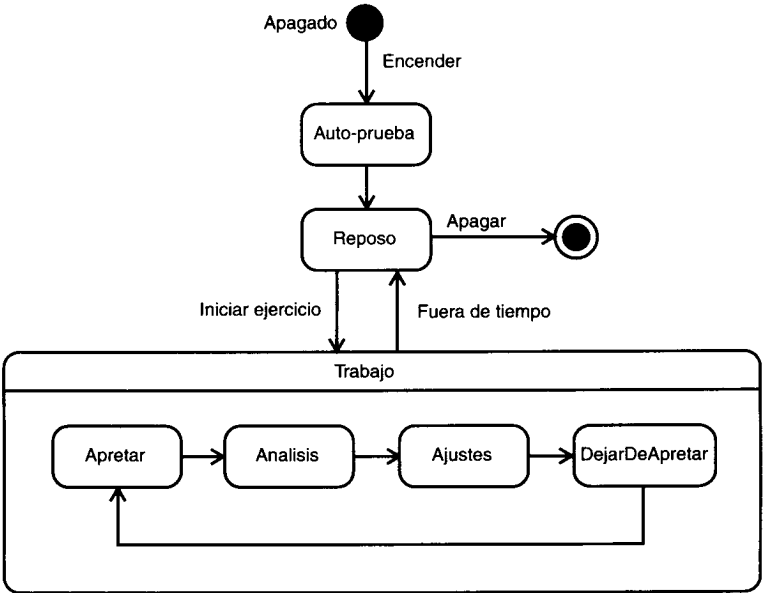
FIGURA 23.12
Diagrama de colaboraciones para “Apretar la barra”.



Cambios de estado generales

Además de los cambios de estado dentro de una interacción, podemos examinar los cambios de estado en todo el sistema. Por lo general, esperamos que TecnoApretón esté en modo de Trabajo o en modo de Reposo (por ejemplo, entre lapsos de un ejercicio). También podría estar en el estado de Apagado. Como podría imaginar, el estado Trabajo es un objeto compuesto. La figura 23.13 muestra los detalles.

FIGURA 23.13
Cambios de estado de TecnoApretón.

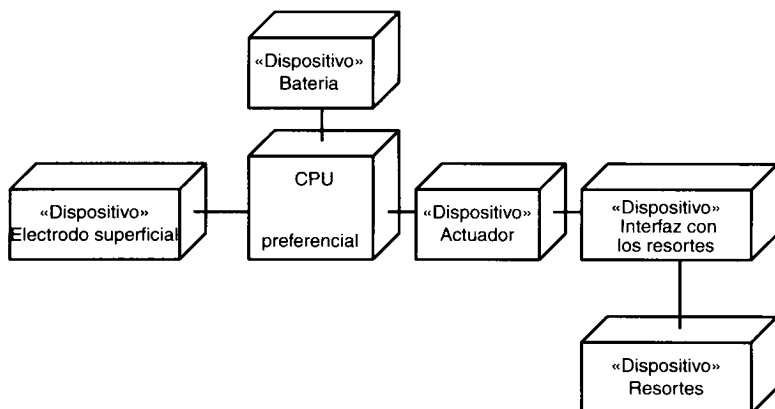


Distribución

¿Qué tal luciría TecnoApretón una vez que se lleve a cabo? La figura 23.14 es un diagrama de distribución que muestra las partes del sistema, junto con una batería que suministra la energía.

FIGURA 23.14

Un diagrama de distribución para TecnoApretón.



Flexiones en sus músculos

Cuando los socios recibieron los diagramas UML de TecnoApretón, empezaron las maquinaciones.

“Éste es un concepto que podríamos expandir”, dijo Goniff.

“¿Cómo?”, preguntó Nar.

“Piénsenlo. ¿Cuántos músculos hay en el cuerpo humano? Podríamos crear un dispositivo ejercitador inteligente que se centre en varios de ellos.”

“Ah, ¿sí?” preguntó nuevamente Nar, cautivado.

“¡Claro!”, dijo Goniff. “Si llevamos el concepto de resortes, electrodos y CPU un poco más allá, podríamos desarrollar una barra con pesas portátil e inteligente que la gente podría llevar consigo cuando viajara. No pesaría mucho, dada la liviandad de los resortes que darían la resistencia y la CPU que daría la inteligencia. Podríamos llamarlo: ‘TecnoPesas’.”

“Sí”, dijo Nar, “o podríamos seguir otro derrotero y hacer máquinas por separado para cada parte del cuerpo”.

“¡Así es! Algo así como ‘TecnoPecho’.”

“O ‘TecnoBrazo’”.

“O ‘TecnoPierna’.”

“¿Qué tal ‘TecnoAsentadera’?”

En este momento, LaHudra ya no podía soportar más.

“Tengo uno para ustedes dos”, les dijo a sus socios.

“¿Cuál?” preguntaron al unísono.

“TecNoSePasen.”

Resumen

Un sistema incrustado es una computadora que se encuentra dentro de algún otro dispositivo, como un electrodoméstico. La programación de un sistema incrustado requiere mucho conocimiento del dispositivo donde se encontrará el sistema. Un sistema incrustado podría ser *tolerante*, significa que no tiene que cumplir con plazos rigurosos, o *estricto*, que sí tiene que cumplir con ellos.

El tiempo, los subprocesos (programas sencillos que son partes de una aplicación) y las interrupciones (dispositivos de hardware que dejan saber a la CPU que ha ocurrido algo) son conceptos importantes en los sistemas incrustados. Un interruptor en particular, el ciclo del reloj, sucede a intervalos regulares y funciona como un latido del corazón del sistema.

Un sistema operativo en tiempo real (RTOS) dirige el tráfico entre subprocesos e interrupciones. El núcleo es la parte del RTOS que administra el tiempo que ocupa la CPU en subprocesos individuales. El programador de tiempo del núcleo determina cuál tarea se ejecutará a continuación. Un núcleo puede ser preferencial (en el que un subproceso de mayor prioridad tiene preferencia sobre uno de baja prioridad cuando finaliza una rutina para el servicio de interrupciones) o cooperativo (en el que la tarea interrumpida continúa después que la rutina para el servicio de interrupciones finaliza).

Estos conceptos los aplicamos mediante el modelado de un inteligente dispositivo ejercitador que varía su resistencia en función de la fuerza con que trabaje un músculo.

Preguntas y respuestas

P Usted habló de sistemas “inteligentes”. ¿Tales sistemas incrustados incluyen algo como la Inteligencia Artificial?

R Así es. Una variedad de la IA, llamada “lógica vagamente definida” o “fuzzy logic”, se encuentra en el corazón de diversos tipos de sistemas incrustados.

P ¿Un tipo de RTOS es más adecuado que otro para cierto tipo de aplicaciones de sistemas incrustados?

R Sí. Un tipo del cual no di mayores explicaciones, el superciclo, es el RTOS más sencillo. Con frecuencia está incrustado en aplicaciones de alto volumen como los juguetes. El núcleo preferencial es el RTOS elegido para sistemas complejos.

Taller

He incrustado algunas preguntas para verificar su nuevo conocimiento, y las respuestas están incrustadas en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Qué es un sistema incrustado?
2. ¿Qué es un evento asincrónico?
3. En términos de sistemas incrustados, ¿qué es un sistema “estricto” y qué un sistema “tolerante”?
4. ¿Qué sucede en un “núcleo preferencial”?

Ejercicios

1. Imagínese un sistema incrustado para un tostador. Asuma que el tostador cuenta con un sensor que analiza una rebanada de pan y su nivel de tueste. Asuma, a su vez, que puede establecer qué tan tostado desea su pan. Dibuje un diagrama de clases para este sistema. Incluya al sensor, la CPU y el elemento calentador (¡y la rebanada de pan!).
2. Dibuje un diagrama de secuencias para el sistema incrustado del tostador. Justifique su elección de un núcleo preferencial o uno cooperativo. Y, aprovechando el viaje, dibuje también un diagrama de distribución.

HORA 24



El futuro del UML

Por último, analizaremos algunas extensiones actuales del UML y algunas extensiones potenciales.

En esta hora se tratarán los siguientes temas:

- Extensiones para los negocios
- Lecciones de las extensiones de negocios
- Interfaces gráficas de usuario
- Sistemas expertos

Hemos llegado a la última hora. Ha sido un largo trecho, pero en el proceso hemos visto mucho del UML. En las últimas dos horas ha visto aplicaciones en áreas innovadoras. En esta hora, veremos un panorama de todo con una extensión actual UML y una mirada a otras áreas de aplicación del UML.

Hemos tratado las extensiones del UML en la hora 14, “Nociones de los fundamentos del UML”. Como lo indicamos entonces, puede extender el UML mediante la adición de estereotipos que le permitan hacer su propio

modelo de su dominio. También podrá agregar estereotipos gráficos que aclaren la información trasladada a su modelo. Los diagramas de distribución son buenos ejemplos de ello porque las figuras con frecuencia sustituyen a los iconos cúbicos del UML.

La meta de esta hora es que empiece a pensar en cómo podría aplicar el UML en su dominio. Como cualquier lenguaje, el UML se encuentra en proceso de evolución. Su futuro depende de cómo lo utilicen y lo extiendan los modeladores.

Extensiones para los negocios

Una extensión popular es un conjunto de estereotipos diseñados para modelar un negocio. Los estereotipos abstraen ciertas ideas primordiales de las que se conforma un negocio. Podrá visualizarlas en términos de símbolos UML que ya conoce o como iconos especializados (creados por el Amigo del UML Ivar Jacobson). La intención es la de modelar situaciones del mundo de los negocios, en lugar de que funcionen como los fundamentos para la construcción del software.

Dentro de un negocio, la clase obvia es *trabajador*. En el contexto de esta extensión

TERMINO NUEVO

UML, un trabajador actúa dentro de los negocios, interactúa con otros trabajadores y participa en los casos de uso. Un trabajador puede ser un *trabajador interno* o un *trabajador eventual*. Un trabajador interno interactúa con otros trabajadores dentro de los negocios, y un eventual lo hace con los actores fuera de los negocios. Una *entidad* no inicia ninguna interacción, pero participa en los casos de uso. Los trabajadores interactúan con las entidades.

La figura 24.1 le muestra la acostumbrada notación UML para estos estereotipos, junto con los iconos especializados. Para cada uno, he incluido un ejemplo del dominio restaurante.

Las extensiones de negocios incluyen a dos estereotipos de asociación: «comunica» y

TERMINO NUEVO

«suscribe». El primero es para las interacciones entre objetos. El segundo describe una asociación entre un origen (llamado *suscriptor*) y un destinatario (llamado *publicador*). El origen establece un conjunto de eventos. Cuando uno de esos eventos sucede en el destinatario, el origen recibe una notificación.

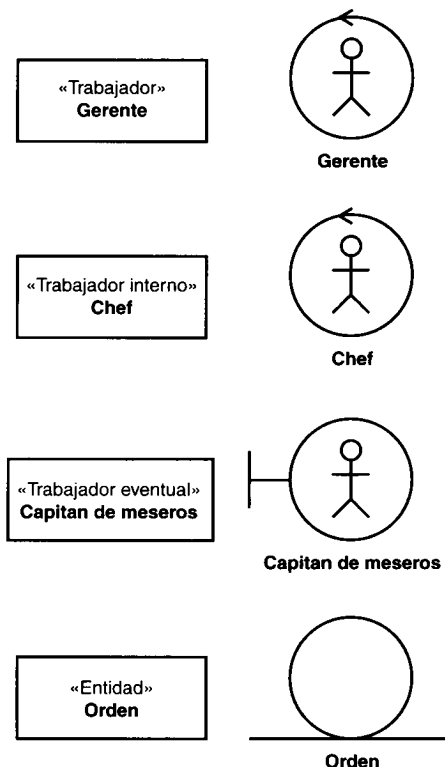
Las entidades se combinan para conformar *unidades de trabajo*, que son conjuntos de

TERMINO NUEVO

objetos orientados a tareas. Las unidades de trabajo, las clases y las asociaciones se combinan para conformar *unidades organizativas*. Una unidad organizativa (que puede incluir a otras unidades organizativas) corresponde a una unidad organizativa del negocio.

FIGURA 24.1

*Esteriotipos para el
modelado de negocios.*



Lecciones de las extensiones de negocios

Las extensiones de negocios nos enseñan algunas lecciones importantes. Para empezar, es evidente que con algo de imaginación es posible utilizar iconos sencillos y representaciones que capturen aspectos fundamentales del dominio. El mundo operativo es “sencillo”. En segundo término, las representaciones nos ayudan a pensar respecto a, y a crear soluciones en, un dominio.

Tomaremos en cuenta tales lecciones conforme exploremos y llevemos al UML a dos importantes proyectos de modelado: las interfaces gráficas de usuario y los sistemas expertos.

Interfaces gráficas de usuario

Un sello de los paquetes contemporáneos de software, la GUI (interfaz gráfica de usuario) ha llegado para quedarse. GRAPPLE y otros procesos y metodologías de desarrollo se aplican a una sesión JAD para la creación de la GUI de una aplicación.

En un documento de diseño, por lo general, incluirá capturas de pantalla para mostrar a sus clientes y desarrolladores cómo lucirá la GUI a los usuarios. Por varias razones, aun podría necesitar un diagrama especializado para modelar una GUI.

Conexiones a casos de uso

La razón primordial tiene que ver con los casos de uso. Como la mayor parte de un proyecto de desarrollo, el desarrollo de la GUI está conducido por casos de uso. De hecho, la GUI se conecta directamente con los casos de uso porque es la ventana por la cual el usuario final iniciará y completará los casos de uso. Podría ser difícil utilizar capturas de pantalla para establecer la relación entre las pantallas y los casos de uso.

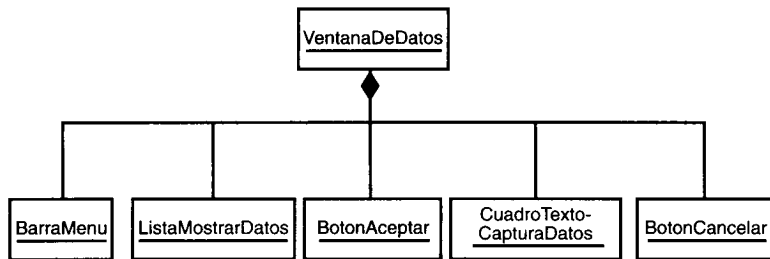
Otra razón es que probablemente necesitemos capturar la evolución en el proceso de razonamiento conforme la GUI tome forma. En GRAPPLE, el desarrollo de la GUI inicia cuando los usuarios finales que participan en la sesión JAD maniobren las notas autoadheribles (que representan controles en la pantalla) en grandes hojas de papel (que representan pantallas). Podría ser útil contar con un tipo de diagrama que capture los resultados de tales maniobras de manera directa (uno que un modelador pueda modificar con facilidad, cuando los participantes de la sesión JAD modifiquen el diseño).

Un diagrama que muestre las conexiones de las pantallas con los casos de uso ayudaría a los que participan en la sesión JAD a recordar lo que se supone que hará cada pantalla cuando coloquen los componentes de la misma. Mostrar las conexiones con los casos de uso también ayudará a asegurar que todos los casos de uso sean instaurados en el diseño final.

Modelado de la GUI

Un modelo típico en UML podría representar a una ventana de una aplicación particular como un objeto compuesto de varios controles, como se ve en la figura 24.2.

FIGURA 24.2
*Un modelo UML
de una ventana.*

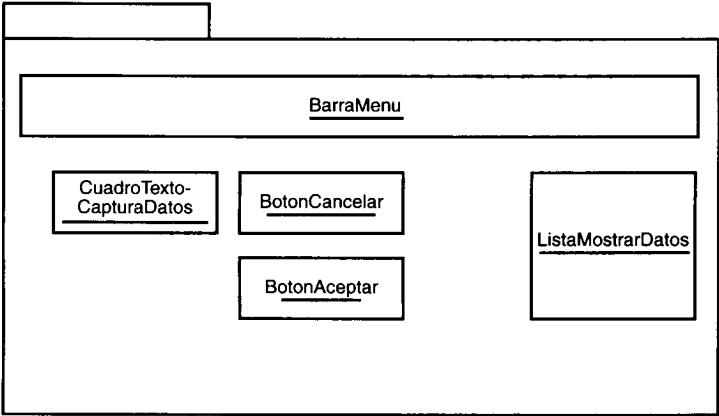


Podríamos utilizar atributos para agregar una ubicación en el espacio de cada componente: una ubicación horizontal y una vertical medida en píxeles. Otro par de atributos podría representar al tamaño del componente (alto y ancho). No obstante, es más fácil abarcar ambos componentes si los visualizamos. Podríamos especificar que un paquete represen-

tará a una ventana, y que la ubicación y tamaño de los objetos dentro del paquete refleje su ubicación y tamaño en la ventana. La figura 24.3 hace lo que indiqué.

FIGURA 24.3

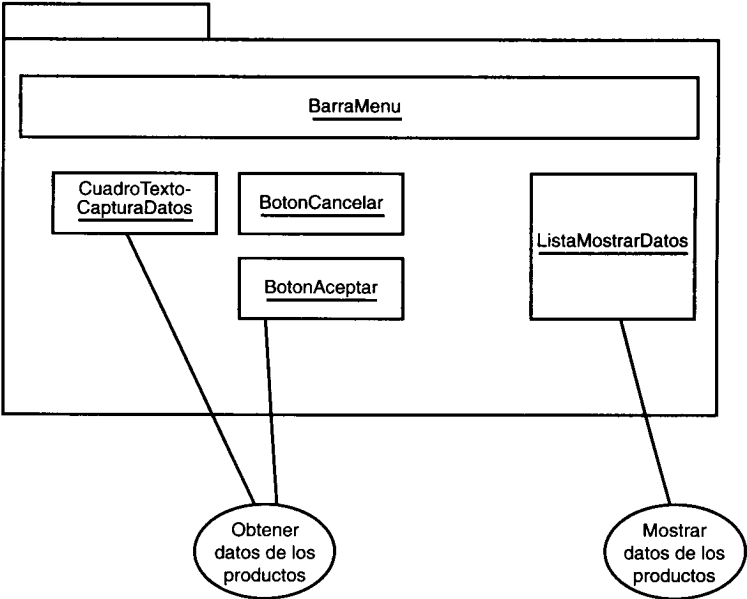
Modelo de una ventana que muestra la ubicación de los componentes.



La figura 24.4 es el diagrama híbrido que agrega el toque final al mostrar las conexiones con los casos de uso.

FIGURA 24.4

Modelado de una ventana que muestra la forma en que los componentes de la pantalla se conectan a los casos de uso.



Este tipo de modelado no evita que se muestren las capturas de pantalla. En lugar de ello, podría ser una adición útil: un diagrama esquemático que se centre en el panorama general.

Sistemas expertos

Los sistemas expertos surgieron a la popularidad en la década de los años ochenta. Cuando aparecieron no fueron más que una curiosidad, pero hoy son parte de la corriente principal del cómputo.

Un sistema experto está diseñado para capturar el conocimiento y experiencia de una persona versada en un dominio específico. Almacena esa experiencia en un programa de cómputo. La intención es utilizar al sistema experto para responder a preguntas repetitivas para que la persona experta no tenga que hacerlo, o almacenar la experiencia para que esté disponible cuando la persona no lo esté.

Componentes de un sistema experto

TÉRMINO NUEVO

La experiencia se encuentra en la *base de conocimientos* del sistema experto, como un conjunto de reglas condicionales *if-then* (si-entonces). La parte *if* de la condición describe ciertas situaciones reales en el dominio del experto. La parte *then* establece las acciones por realizar en tal situación. ¿De qué manera se registra la experiencia en la base de conocimientos? Un *ingeniero de conocimiento* realiza extensas entrevistas con un experto, graba los resultados y los presenta en software. Es similar a la entrevista que se realiza en un análisis de dominio, aunque las sesiones para la ingeniería del conocimiento son, por lo general, más extensas.

La base de conocimientos no es el único componente en un sistema experto. Si lo fuera, un sistema experto podría ser tan sólo una lista de reglas condicionales *if-then*. Lo que se necesita es un mecanismo para operar a lo largo de la base de conocimientos para resolver un problema. A tal mecanismo se le conoce como *motor de deducciones*. Otra pieza necesaria del rompecabezas es un *área de trabajo* que contenga las condiciones de un problema que el sistema tenga que resolver. Claro está que otro componente más es la interfaz del usuario para indicar las condiciones del problema. La captura de condiciones podría realizarse mediante una lista de verificación, preguntas y respuestas de opción múltiple o un sistema extremadamente sofisticado sentado en el idioma natural. La figura 24.5 le muestra un diagrama de clases UML de un sistema experto.

Para interactuar con un sistema experto, el usuario captura las condiciones de un problema en la interfaz del usuario, misma que las almacena en el área de trabajo. El motor de deducciones se vale de tales condiciones para buscar una solución en la base de conocimientos. La figura 24.6 presenta un diagrama de secuencias para este proceso.

FIGURA 24.5
Diagrama de clases de un sistema experto.

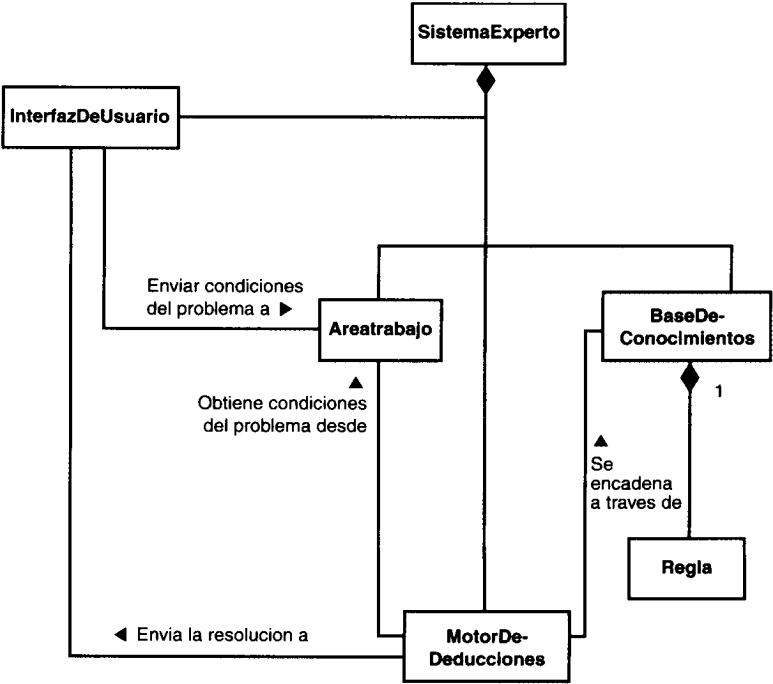
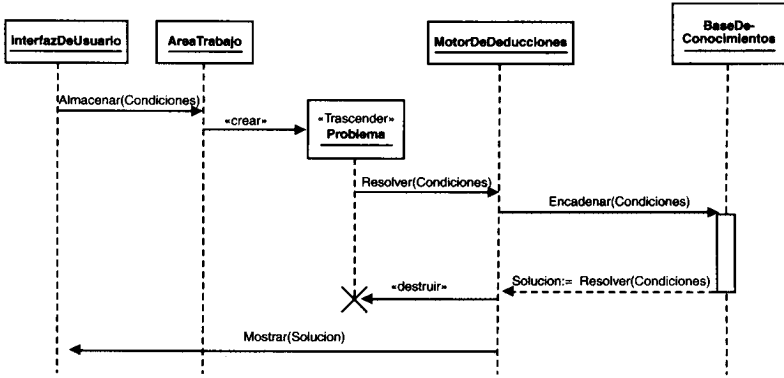


FIGURA 24.6
Interacción con un sistema experto.



El área de trabajo podría equipararse a la memoria temporal (datos en caché), en tanto que la base de conocimientos sería la memoria permanente (datos en disco) y el motor de deducciones sería el proceso para resolver el problema.

Un ejemplo

Un motor de deducciones por lo general analiza su base de conocimientos en una de dos formas, y la mejor manera de explicarlo es mediante un ejemplo. Suponga que tenemos un sistema experto que captura la experiencia de un fontanero. Si uno de sus grifos gotea, utilizaría al sistema experto y le explicaría el problema. El motor de deducciones haría el resto.

Dos de las reglas en su base de conocimientos podrían lucir así:

Regla 1:

SI tiene un grifo que gotea

Y el grifo es de compuerta

Y la gotera está en la manija

ENTONCES ajuste el empaque

Regla 2:

SI el empaque está ajustado

Y la gotera persiste

ENTONCES reemplace el empaque

Sin profundizar en el mundo de la fontanería, bastará decir que estas dos reglas están relacionadas (obviamente): observe la similitud entre la parte then de la Regla 1 y la parte if de la Regla 2. Tal similitud es el fundamento para analizar la base de conocimientos (la cual tiene mucho más que dos reglas). El motor de referencia podría iniciar con una solución potencial como “reemplazar el empaque” en la Regla 2, e ir en retrospectiva para ver los detalles del problema que demanda tal solución.

¿De qué manera el motor de deducciones “va en retrospectiva”? Analiza la parte if de la regla que tenga la solución y busca una regla cuya parte then concuerde. En nuestro ejemplo de dos reglas, eso es fácil: la Regla 1 tiene una parte then coincidente. En las aplicaciones de tono empresarial, esto no es tan sencillo debido a que una base de conocimientos podría almacenar cientos, tal vez miles, de reglas.

Después de que el motor de deducciones haya encontrado una regla, verifica si las condiciones de la parte if concuerdan con las condiciones del problema. Si es así, el motor continúa su búsqueda en la misma dirección: una parte then, verifica la parte if, otra parte then que concuerde y así sucesivamente. Cuando el motor de deducciones se queda sin reglas, le pide más información al usuario. La meta de ello es que si el trayecto por las reglas es exitoso (esto es, que concuerde con las condiciones del problema), el sistema ofrecerá la solución potencial original como la solución al problema. Si no es el caso, el sistema intentará un nuevo trayecto.

TERMINO NUEVO

Esta técnica de probar una solución y verificar si las condiciones asociadas concuerdan con las condiciones del problema se conoce como *encadenado regresivo*; “regresivo” porque inicia con las partes then y procede a examinar las partes if.

TERMINO NUEVO

Como podrá imaginar, otra técnica inicia con las partes if y corresponde con las partes then, y se conoce como *encadenado progresivo*. Así es como trabaja. El usuario establece las condiciones del problema y el motor de deducciones localiza una regla cuya parte if concuerde con las condiciones. Verifica la parte then y busca una regla cuya parte if concuerde con tal parte then. En nuestro ejemplo, suponga que la parte if de la Regla 1 concuerde con las condiciones del problema. El motor de deducciones verifica la parte then de la Regla 1 y, luego, busca una regla con una parte if que concuerde. Nuevamente, esto es sencillo en nuestro caso con sólo dos reglas. Cuando el sistema se queda sin reglas por comparar, ofrece las partes then de la regla final como la solución. La parte “progresiva” del “encadenado progresivo” se refiere a este movimiento de las partes if a las partes then.

Si tuviéramos que modelar un sistema experto como en la figura 24.5, sería de mucha ayuda agregar un estereotipo que indique el tipo de encadenamiento que realice el motor de deducciones. Agregaríamos ya sea «encadenamiento progresivo» o «encadenamiento regresivo» a la clase compuesta SistemaExperto.



Ambos tipos de encadenamiento son ejemplos del patrón de diseño Cadena de responsabilidades que vio anteriormente. En cada uno, el sistema busca a un “sucesor” de la regla.

Así como en ocasiones la Cadena de responsabilidades finaliza sin encontrar a un sucesor, un sistema experto tampoco le presentará siempre una solución.

Modelado de la base de conocimientos

¿Qué es lo que puede agregar el UML a todo esto, y para qué lo querríamos? Uno de los puntos primordiales en el desarrollo de un sistema experto es la ausencia de una norma sólida para generar una representación visual de las reglas de la base de conocimientos. Una representación basada en el UML podría requerir de un largo proceso para

estandarizar el campo que propenda a buenas prácticas de documentación. No es suficiente que el conocimiento se encuentre en una representación informática dentro de una base de conocimientos: las reglas deberían estar también en un documento.

¿Cómo podríamos representar a una base de conocimientos bajo el espíritu del UML? Es evidente que una posibilidad sería la de representar cada regla como un objeto. Contar con uno de los atributos para que sea la parte if, otro la parte then y agregar otros atributos conforme sea necesario. La figura 24.7 le muestra este diseño.

FIGURA 24.7
Representación de las reglas como objetos.

<u>Regla 1</u>
partelf = grifo que gotea Y grifo de compuerta Y gotera en manija parteThen = Ajustar empaque

<u>Regla 2</u>
partelf = empaque ajustado Y aun hay gotera parteThen = cambiar empaque

Aunque esto es eminentemente factible (y muchos desarrolladores lo hacen), creo que las reglas son muy importantes para garantizar su propia representación. Además de ser el fundamento de las bases de conocimientos, el creciente énfasis en la administración del conocimiento dentro de las organizaciones e instituciones espera algo único.

¿Cómo podría lucir tal representación única? Para empezar, necesitamos asegurarnos que mostraremos algo que dé el contenido de la parte if de la regla, y el de la parte then. Para hacer útil a esta representación, también necesitamos algo para visualizar las conexiones entre las reglas.

Esto podría dificultarse con rapidez. Las reglas de talla industrial tienden a contar con mayor información que las dos reglas de fontanería que le mostré, y las reglas se inclinan a proliferar. Tenemos que balancear la proliferación respecto a la necesidad de la simplicidad.

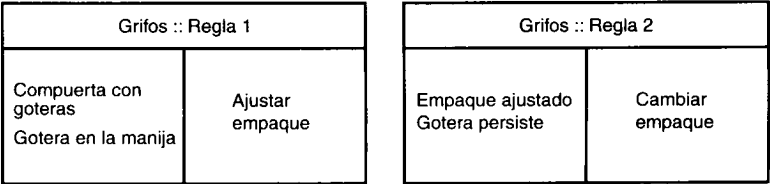
Primero cree un sencillo icono que represente a la regla. Empezaremos con un cuadro dividido a la mitad por una línea vertical. La mitad izquierda representará a la parte if y la derecha a la parte then. Dentro de cada parte escribiremos un resumen significativo del contenido. La figura 24.8 le muestra lo que le quiero decir, con el ejemplo de las dos reglas de fontanería.

FIGURA 24.8
Las dos reglas de fontanería moldeadas en una representación visual.

Compuerta con goteras Gotera en la manija	Ajustar empaque	Empaque ajustado Gotera persiste	Cambiar empaque
--	-----------------	-------------------------------------	-----------------

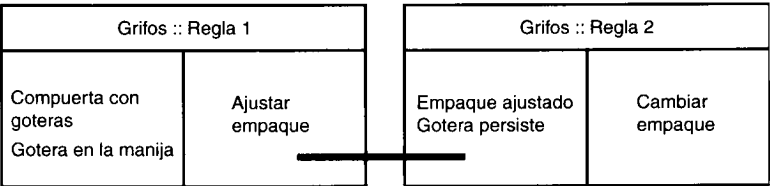
Ahora tenemos que incorporar cierta información de identificación para cada regla. Agreguemos un compartimiento en la parte superior de cada cuadro que contenga un identificador numérico. Esto logra un par de cosas: (1) Convierte a cada regla en única, y (2) muestra a dónde ir en un catálogo de reglas para obtener una descripción y explicación completa de la regla. Si una regla es parte de un subgrupo de reglas (como en un subconjunto “grifo” de nuestra base de conocimiento de fontanería), podemos tratar al subgrupo como un paquete. Luego, agregar el paquete de información al identificador de la forma usual propia del UML: hacer que el nombre del paquete anteceda a un par de dos puntos (::), que antecedan al identificador. La figura 24.9 muestra tal adición.

FIGURA 24.9
Adición de un identificador a cada regla.



Ahora, representaremos la relación entre los dos como una línea entre la parte then de la Regla 1 y la parte if de la Regla 2. La figura 24.10 le muestra la conexión.

FIGURA 24.10
Conexión de la parte then de una regla a la parte if de la otra.

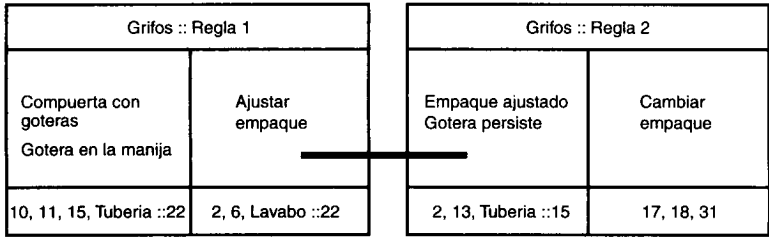


A diferencia de los dos conjuntos de pares de reglas, una regla en un sistema experto verdadero se relaciona usualmente a más de una regla distinta. Si las reglas relacionadas no están próximas —ya sea en la base de conocimientos o en la documentación— será útil tener una forma de mostrar la relación aún cuando no podamos contar con líneas de conexión.

Los compartimientos de la parte baja del icono nos permitirán lograrlo. Si los colocamos por debajo de los que ya tenemos, podremos mostrar identificadores para otras reglas, como en la figura 24.11. El compartimiento inferior de la izquierda identifica a las reglas cuyas partes then se conectan con la parte if de ésta. El compartimiento inferior de la derecha identifica reglas cuyas partes if se conectan con la parte then de ésta.

FIGURA 24.11

Los compartimientos en la parte inferior del icono de la regla identifican a las reglas relacionadas.



Como en el caso de los diagramas de clases, los compartimientos dentro del icono de la regla podrían ser omitidos de acuerdo con el propósito del diagrama. La idea es mostrar de manera concisa las conexiones entre las reglas, así como su contenido, y con ello comunicar con claridad la naturaleza de la base de conocimientos.

El modelo del sistema experto es más “drástico” que el de la GUI, en el sentido de que propone un nuevo “elemento de perspectiva” (el icono de regla) para el UML. El modelo para la GUI, por otra parte, es un diagrama híbrido que consta de elementos UML actuales.

Eso es todo, amigos

Hemos llegado al final del camino. Ahora que cuenta con todo un equipaje lleno de trucos del UML, ya está listo para continuar por sí mismo y aplicarlos a su dominio. Verá que conforme obtenga experiencia, hará adiciones a tal equipaje. Posiblemente tenga algunas sugerencias por agregar al UML. Si es su caso, continuará con una gran tradición.

A principios del siglo XX, el renombrado matemático Alfred North Whitehead apuntó la importancia de los símbolos y de su uso. Un símbolo, como indicó, representa a una idea: la importancia de un símbolo es que rápida y concisamente muestra la forma en que una idea encaja en un complejo grupo de ideas diversas.

Al principio de un nuevo milenio, las observaciones de Whitehead aún se aplican en el mundo del desarrollo de sistemas. Los símbolos cuidadosamente diseñados nos muestran el proceso del raciocinio y las complejidades que hay detrás de los maravillosos sistemas que intentamos generar, y nos ayudan a asegurar su rendimiento eficiente cuando los generemos.

Resumen

Conforme los modeladores extiendan y moldeen al UML para cumplir con sus necesidades, modelarán su propio futuro. En esta hora, hemos visto una extensión para el modelado de los negocios y sugerí algunas formas de aplicar el UML en otras áreas.

Como lección de la simplicidad de las extensiones de negocios, hemos explorado las formas de modelar las GUI y los sistemas expertos. Para modelar una GUI, establecemos un diagrama híbrido que muestre las relaciones de espacio de los componentes de la pantalla, y que muestre sus conexiones y casos de uso. Esto tiene la ventaja de mostrar la evolución de una GUI conforme toma forma, y mantiene a los casos de uso correspondientes en el centro de la atención.

En un sistema experto, las reglas de condiciones (if-then) son el bloque de construcción de una base de conocimientos, el componente que contiene el conocimiento de un experto en algún dominio humano. Sugerimos un diagrama que visualice las reglas y sus relaciones internas. En este diagrama, un cuadro dividido en compartimientos modela a la regla. Un compartimiento contiene al identificador de la regla, otro resume la parte if, otro la parte then y otras dos muestran las reglas relacionadas. Los vínculos a las reglas adyacentes aparecen como líneas de conexión entre las partes adecuadas de las reglas.

Preguntas y respuestas

- P Aunque en principio parecería que un sistema experto no es difícil de modelar, parece que podría convertirse en un programa extremadamente difícil de escribir.**
- R** Lo será si tiene que crear uno desde el inicio. Por fortuna, la mayor parte de la programación la realiza usted en un paquete llamado *shell* para un sistema experto. Todos los componentes ya están hechos. Tan sólo deberá agregar el conocimiento. No obstante, extraer el conocimiento de una persona experta no siempre es una tarea fácil.
- P ¿Qué los vendedores de shells para sistemas expertos no tienen una notación para representar a las reglas?**
- R** Sí, y allí está el problema. No hay una notación que sea la estándar.

Taller

Las preguntas de este taller verificarán su conocimiento en la aplicación del UML a las GUI y a los sistemas expertos. Las respuestas a los cuestionarios podrán encontrarse en el Apéndice A, “Respuestas a los cuestionarios”.

Cuestionario

1. ¿Cuáles son las ventajas de nuestro modelo de una GUI?
2. ¿Cuáles son los componentes de un sistema experto?
3. ¿Qué características de un sistema experto comprende nuestro diagrama?