5. DIAGRAMAS DE ESTADO

5.1. INTRODUCCIÓN

Además de la estructura estática y del comportamiento dinámico, las vistas funcionales se pueden utilizar para describir a los sistemas. Las vistas funcionales ilustran la funcionalidad que proporciona un sistema. Los casos de uso son las descripciones funcionales del sistema. Normalmente, son modelados en la etapa de análisis de requisitos para describir y capturar cómo los actores podrían utilizar un sistema. Los diagramas de casos de uso deberían capturar solamente cómo un actor puede usar un sistema, pero no cómo debe ser construido dicho sistema.

Las clases y las interacciones implementan los casos de uso en el sistema. Las interacciones son expresadas en diagramas de secuencia y/o colaboración. Entonces hay un enlace entre la visión funcional y la visión dinámica del sistema. Las clases utilizadas en la implementación de los casos de uso son modeladas y descritas en los diagramas de clase, en los diagramas de estado y/o actividad.

5.2. DIAGRAMAS DE ESTADO

Los diagramas de estado muestran el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación en respuesta a eventos (por ejemplo, mensajes recibidos, tiempo rebasado o errores), junto con sus respuestas y acciones. También ilustran qué eventos pueden cambiar el estado de los objetos de la clase. Normalmente contienen: estados y transiciones. Como los estados y las transiciones incluyen, a su vez, eventos, acciones y actividades, vamos a ver primero sus definiciones.

Al igual que otros diagramas, en los diagramas de estado pueden aparecer notas explicativas y restricciones.

5.2.1. Eventos

Un evento es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una:

- condición que toma el valor de verdadero (normalmente descrita como una expresión booleana). Es un EventoCambio.
- recepción de una señal explícita de un objeto a otro. Es un EventoSeñal.
- recepción de una llamada a una operación. Es un EventoLlamada.
- paso de cierto período de tiempo, después de entrar al estado actual, o de cierta hora y fecha concretas. Es un EventoTiempo.

El nombre de un evento tiene alcance dentro del paquete en el cual está definido y puede ser usado en los diagramas de estado por las clases que tienen visibilidad dentro del paquete. Un evento no es local a la clase donde está declarado.

5.2.2. Acciones

Una acción es una operación atómica, que no se puede interrumpir por un evento y que se ejecuta hasta su finalización. Una acción puede ser:

- una llamada a una operación (al objeto al cual pertenece el diagrama de estado o también a otro objeto visible),
- la creación o la destrucción de otro objeto,
- el envío de una señal a un objeto.

5.2.3. Actividades

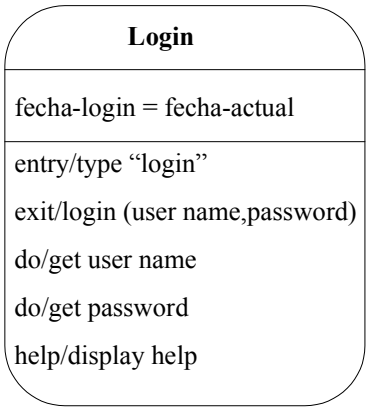
Cuando un objeto está en un estado, generalmente está esperando a que suceda algún evento. Sin embargo, a veces, queremos modelar una actividad que se está ejecutando. Es decir, mientras un objeto está en un estado, dicho objeto realiza un trabajo que continuará hasta que sea interrumpido por un evento.

Por lo tanto, una acción contrasta con una actividad, ya que ésta última puede ser interrumpida por otros eventos.

5.2.4. Estados

Un estado identifica una condición o una situación en la vida de un objeto durante la cual satisface alguna condición, ejecuta alguna actividad o espera que suceda algún evento. Un objeto permanece en un estado durante un tiempo finito (no instantáneo).

Un estado se representa gráficamente por medio de un rectángulo con los bordes redondeados y con tres divisiones internas. Los tres compartimentos alojan el nombre del estado, el valor característico de los atributos del objeto en ese estado y las acciones que se realizan en ese estado, respectivamente. En muchos diagramas se omiten los dos compartimentos inferiores.



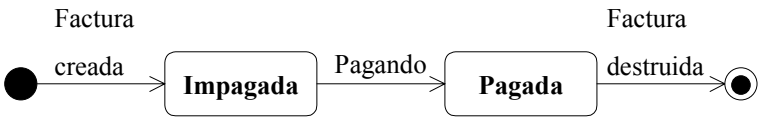


Figura 5.1

En el primer ejemplo de la Figura 5.1 viene representado el estado Login junto con sus tres divisiones. Asimismo, los diagramas de estado tienen un punto de comienzo, el estado inicial, que se dibuja mediante un círculo sólido relleno, y un (o varios) punto de finalización, el estado final, que se dibuja por medio de un círculo conteniendo otro más pequeño y relleno (es como un ojo de toro). Dichos estados, inicial y final, aparecen marcados en el segundo ejemplo de la Figura 5.1.

5.2.1.1. Compartimento del nombre

Cada estado debe tener un nombre.

En el primer ejemplo de la Figura 5.1 tenemos el nombre de estado: Login, mientras que en el segundo ejemplo hay dos nombres de estado: Impagada y Pagada.

5.2.1.2. Compartimento de la lista de variables

El segundo compartimento es el compartimento de las variables de estado, donde los atributos (variables) pueden ser listados y asignados. Los atributos son aquellos de la clase visualizados por el diagrama de estado.

En el primer ejemplo de la Figura 5.1 tenemos la variable de estado: fecha-login, a la cual se le ha asignado el valor de la fecha del día.

5.2.1.3. Compartimento de la lista de acciones

El tercer compartimento es el compartimento de las transiciones internas, donde se listan las actividades o las acciones internas ejecutadas en respuesta a los eventos recibidos mientras el objeto está en un estado, sin cambiar de estado. La sintaxis formal dentro de este compartimento es:

nombre-evento ‘(’lista-argumentos‘)’ ‘[’guard-condition‘]’ ‘/’ expresión-acción

Las siguientes acciones especiales tienen el mismo formato, pero representan palabras reservadas que no se pueden utilizar para nombres de eventos:

entry ‘/’ expresión-acción

exit ‘/’ expresión-acción

Las acciones entry y exit no tienen argumentos, pues están implícitos en ellas. Cuando se entra al estado o se sale del estado, se ejecuta la acción atómica especificada en expresión-acción.

La siguiente palabra clave representa la llamada de una actividad:

do ‘/’ expresión-acción

La transición especial do nos sirve para especificar una actividad que se ejecuta mientras se está en un estado, por ejemplo, enviando un mensaje,

esperando o calculando. Dicha actividad es la que aparece en expresión-acción.

Por último, las transiciones internas, que son esencialmente interrupciones, se especifican teniendo en cuenta la sintaxis formal. Por ejemplo:

help / display help

La transición interna help representa un evento que no causa ningún cambio de estado, pues sólo muestra, en cualquier momento, una ayuda al usuario que viene expresada en display help.

En el primer ejemplo de la Figura 5.1 aparecen todos los tipos de eventos especificados anteriormente, junto con las acciones y actividades que se realizan al entrar, salir o estar en el estado Login.

5.2.5. Transiciones simples

Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones, cuando un evento ocurre y si ciertas condiciones son satisfechas.

Una transición simple se representa gráficamente como una línea continua dirigida desde el estado origen (*source*) hasta el estado destino (*target*). Puede venir acompañada por un texto con el siguiente formato:

nombre-evento '('lista-argumentos')' '['guard-condition']' '/' expresión-acción '^' claúsula-envío

donde nombre-evento y lista-argumentos describen el evento que da lugar a la transición y forman lo que se denomina event-signature,

donde guard-condition es una condición (expresión booleana) adicional al evento y necesaria para que la transición ocurra.

Si la guard-condition se combina con una event-signature, entonces para que la transición se dispare tienen que suceder dos cosas: debe ocurrir el evento y la condición booleana debe ser verdadera.

donde expresión-acción es una expresión procedimental que se ejecuta cuando se dispara la transición.

Es posible tener una o varias expresión-acción en una transición de estado, las cuales se delimitan con el carácter "/".

donde claúsula-envío es una acción adicional que se ejecuta con el cambio de estado, por ejemplo, el envío de eventos a otros paquetes o clases.

Este tipo especial de acción tiene una sintaxis explícita para enviar un mensaje durante la transición entre dos estados. La sintaxis consiste de una expresión de destino y de un nombre de evento, separados por un punto.

En la Figura 5.2 tenemos un diagrama de estado para un ascensor, donde se combinan los estados con las transiciones simples.

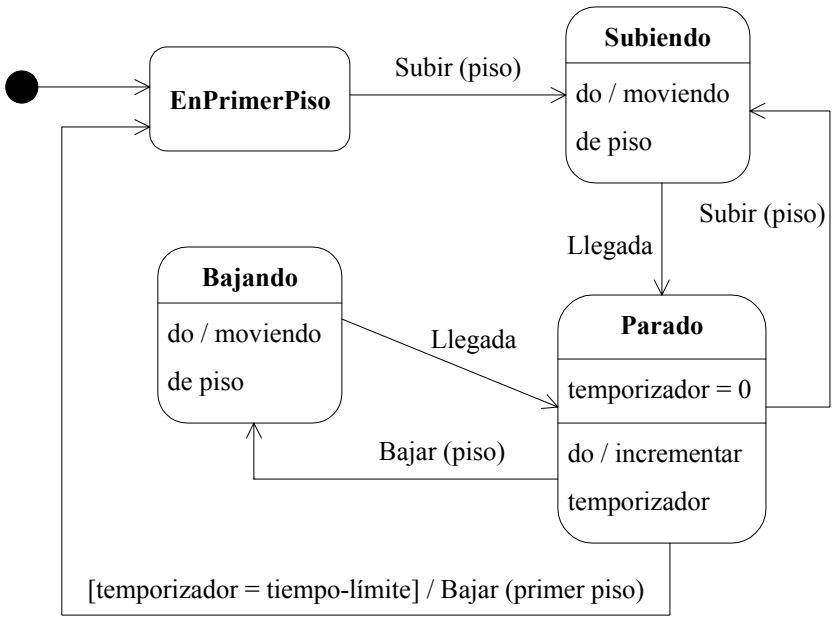


Figura 5.2

El ascensor empieza estando en el primer piso. Puede subir o bajar. Si el ascensor está parado en un piso, ocurre un evento de tiempo rebasado después de un período de tiempo y el ascensor baja al primer piso. Este diagrama de estado no tiene un punto de finalización (estado final).

El evento de la transición entre los estados EnPrimerPiso y Subiendo tiene un argumento, piso (el tipo de este parámetro ha sido suprimido). Lo mismo sucede con los eventos de las transiciones entre Parado y Subiendo y entre Parado y Bajando.

El estado Parado (*Idle state*) asigna el valor cero al atributo temporizador, luego lo incrementa continuamente hasta que ocurra el evento Bajar (piso) o el evento Subir (piso) o hasta que la guard-condition [temporizador = tiempo-límite] se convierta en verdadera.

La transición de estado entre Parado y EnPrimerPiso tiene una guard-condition y una expresión-acción. Cuando el atributo temporizador es equivalente a la constante tiempo-límite, se ejecuta la acción Bajar (primerpiso) y el estado del ascensor cambia de Parado a EnPrimerPiso. Esta transición de estado

[temporizador = tiempo-límite] / Bajar (primerpiso)

se puede convertir en una claúsula-envío tal como:

[temporizador = tiempo-límite] ^ Self.Bajar (primerpiso)

donde la expresión destino es, en este caso, el propio objeto que se evalúa a sí mismo, y el nombre del evento es Bajar (primerpiso), evento significativo al objeto contenido en la expresión destino.

5.2.6. Estados avanzados

Las características de los estados y de las transiciones, vistas en los apartados anteriores, resuelven un gran número de problemas a la hora de modelar un diagrama de estado. Sin embargo, hay otra característica de las máquinas de estado de UML, los subestados, que nos ayudan a simplificar el modelado de aquellos comportamientos complejos.

Un estado simple es aquel que no tiene estructura. Un estado que tiene subestados, es decir, estados anidados, se denomina estado compuesto. Un estado compuesto puede contener bien subestados secuenciales (disjuntos) o bien subestados concurrentes (ortogonales).

5.2.6.1. Subestados secuenciales

Consideremos el problema de modelar el comportamiento de un Cajero Automático (CA). Hay tres estado básicos en los que podría estar este sistema: Parado (esperando la interacción del usuario), Activo (gestionando una transacción del cliente) y Mantenimiento (teniendo que actualizar el efectivo almacenado). Mientras está en Activo, el comportamiento de CA sigue un camino sencillo: validar al cliente, seleccionar una transacción, procesarla e imprimir un recibo. Después de la impresión, el CA vuelve al estado Parado (*Idle*). Podríamos representar estos estados de comportamiento como los estados Validando, Seleccionando, Procesando e Imprimiendo. Incluso sería deseable permitir al cliente seleccionar y procesar múltiples transacciones después de Validando la cuenta bancaria y antes de Imprimiendo un recibo final.

El problema que se nos plantea aquí es que, en cualquier etapa de este comportamiento, el cliente podría decidir cancelar la transacción, volviendo el CA al estado Idle. Usando las máquinas de estado simples, podríamos conseguir dicho efecto, pero es bastante lioso. Como el usuario podría cancelar la transacción en cualquier punto, tendríamos que incluir una transacción de forma correcta desde cada estado de la secuencia Activo. Esto es complicado porque es fácil olvidar incluir estas transacciones en todos los lugares apropiados.

Utilizando los subestados secuenciales, se puede resolver de una forma más sencilla este problema, tal como muestra la Figura 5.3. Aquí, el estado Activo tiene una subestructura, conteniendo los subestados Validando, Seleccionando, Procesando e Imprimiendo. El estado del CA cambia de Parado a Activo cuando el cliente introduce una tarjeta de crédito en el cajero. Al entrar al estado Activo, se ejecuta la acción: leer tarjeta, que viene especificada dentro de dicho estado. Empezando con el estado inicial de la subestructura, el control pasa al estado Validando, después al estado Seleccionando y luego al estado Procesando. Después de Procesando, el control puede regresar a Seleccionando (si el usuario ha seleccionado otra transacción) o puede ir a Imprimiendo. Después de Imprimiendo, hay una transición de vuelta, sin evento ni

condiciones de disparo (*trigger*), al estado Parado. Hay que fijarse que el estado Activo tiene una acción de salida, exit, la cual devuelve la tarjeta de crédito al cliente.

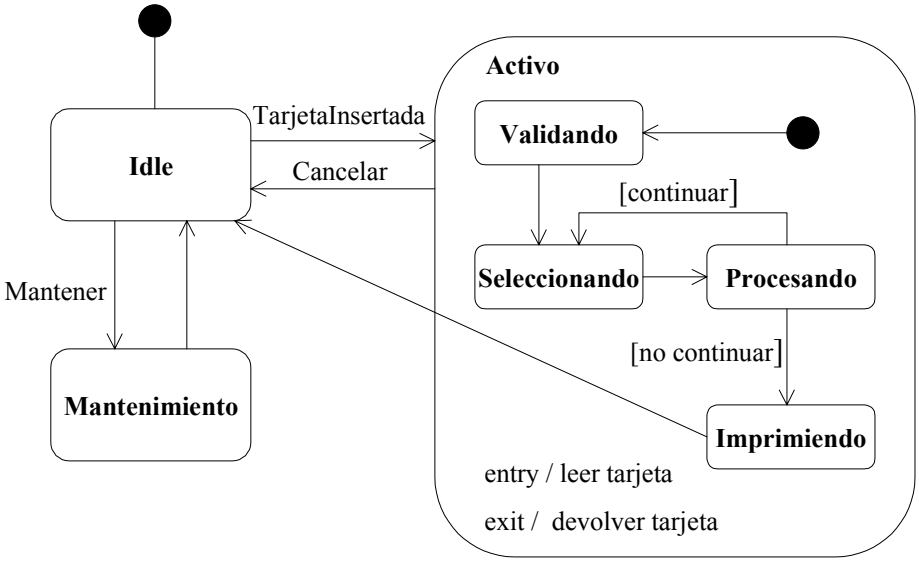


Figura 5.3

La transición del estado Activo al estado Idle ha sido disparada por el evento Cancelar. En cualquier subestado de Activo, el cliente podría cancelar la transacción y que el CA volviese al estado Parado (pero sólo después de devolver la tarjeta de crédito al cliente, ya que la acción exit es enviada al dejar el estado Activo, y no importa qué causó una transición fuera de este estado). Sin subestados, necesitaríamos una transición disparada por Cancelar en cada estado de la subestructura.

Los subestados tales como Validando y Procesando se llaman subestados secuenciales o disjuntos. Dado un conjunto de subestados disjuntos encerrados en un estado compuesto, se dice que el objeto está en el estado compuesto y en uno sólo de sus subestados a la vez. Por lo tanto, los subestados secuenciales dividen el espacio del estado compuesto en estados disjuntos.

Desde fuera de un estado compuesto, una transición puede tener como meta el estado compuesto o como meta un subestado. Si su objetivo es el estado compuesto, la máquina de estado anidada (o lo que es lo mismo, el diagrama de estado con subestados) tiene que incluir un estado inicial, al cual pasa el control después de entrar al estado compuesto y después de ejecutar la acción entry (si la hay). Si su objetivo es un subestado del estado compuesto, el control pasa a dicho subestado, después de ejecutar la acción entry (si la hay) del estado compuesto y luego la acción entry (si la hay) del subestado.

Una transición al salir de un estado compuesto puede tener como su origen el estado compuesto o un subestado del mismo. En ambos casos, el control primero deja el estado anidado (y su acción exit, si la hay, es ejecutada), luego deja el estado compuesto (y su

acción exit, si la hay, es ejecutada). Una transición cuyo origen es el estado compuesto esencialmente interrumpe la actividad de la máquina de estado anidada.

Nota:
1. Una máquina de estado secuencial anidada puede tener un estado inicial y un estado final.

5.2.6.2. Subestados concurrentes

Los subestados secuenciales son el tipo más común de máquina de estado anidada. Sin embargo, en ciertas situaciones de modelado tenemos que especificar subestados concurrentes. Estos subestados nos permiten especificar dos o más máquinas de estado que se ejecutan en paralelo.

La Figura 5.4 nos muestra una expansión del estado Mantenimiento, que aparece en la Figura 5.3.

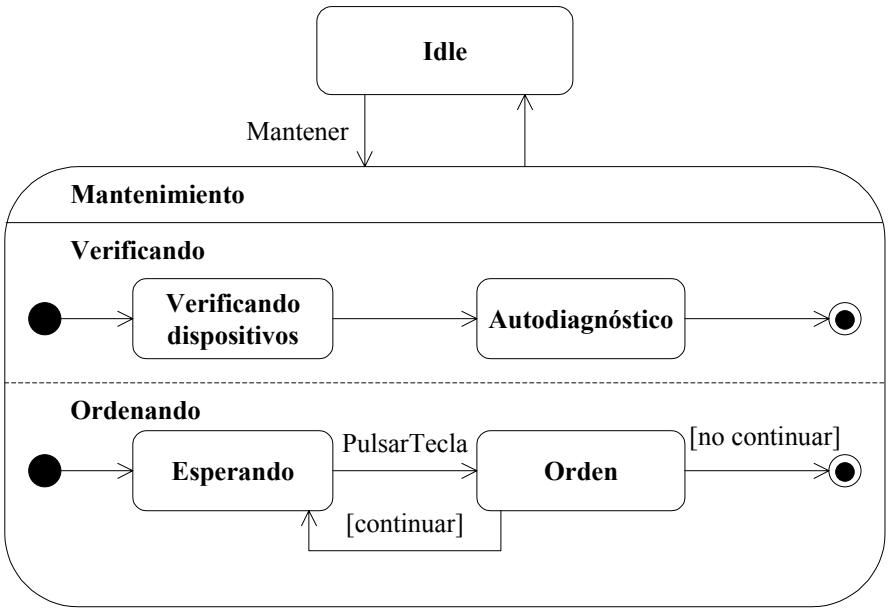


Figura 5.4

El estado de Mantenimiento está descompuesto en dos subestados concurrentes, Verificando y Ordenando, que se encuentran anidados en dicho estado pero separados por una línea discontinua. Cada uno de estos subestados concurrentes a su vez está descompuesto en subestados secuenciales. Cuando el control pasa del estado Parado al estado Mantenimiento, el control se bifurca en dos flujos concurrentes y el objeto implicado estará en el estado Verificando y también en el estado Ordenando. Mientras se encuentre en el estado Ordenando, el objeto estará bien Esperando o bien en el estado Orden.

La ejecución de estos dos subestados concurrentes continúa en paralelo. Luego cada máquina de estado anidada alcanza su final. Si un subestado concurrente alcanza su final antes que el otro, el control en dicho estado espera a su estado final. Cuando ambas

máquinas de estado anidadas alcanzan sus estados finales, el control de los dos subestados concurrentes se juntan de nuevo en un único flujo.

Nota:
1. Una máquina de estado concurrente anidada no tiene un estado inicial y un estado final. Sin embargo, los subestados secuenciales que componen un estado concurrente pueden tener dichos estados.

5.3. DIAGRAMAS DE ACTIVIDAD

Los diagramas de actividad son uno de los cinco diagramas en el UML para modelar los aspectos dinámicos de un sistema. Un diagrama de actividad es un caso especial de un diagrama de estado, en el cual casi todos los estados son estados de acción (identifican qué acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la acción ejecutada en el estado anterior. Este diagrama es el menos familiar y se utiliza poco en UML.

Los diagramas de actividad pueden visualizar, especificar y documentar la dinámica de un conjunto de objetos. También se pueden usar para modelar el flujo de control de una operación. Mientras que los diagramas de interacción enfatizan el flujo de control de un objeto a otro, los diagramas de actividad subrayan el flujo de control de una actividad a otra.

La gran desventaja de los diagramas de actividad es que no indican de forma explícita qué objetos ejecutan qué actividades ni tampoco la forma en que el servicio de mensajería trabaja entre ellos. Para mostrar tales interacciones de forma clara son necesarios los diagramas de interacción, los cuales son más utilizados en la práctica.

Nota:
1. Los diagramas de actividad son útiles cuando queremos describir un comportamiento paralelo, o cuando queremos mostrar qué comportamientos interactúan en varios casos de uso.
2. Los diagramas de interacción se utilizan cuando se quiere mostrar cómo colaboran los objetos para implementar un diagrama de actividad.
3. Los diagramas de estado se utilizan cuando se quiere mostrar cómo cambia un objeto a lo largo de su tiempo de vida.