5.1. Presentación del capítulo

En este capítulo tratamos algunos aspectos avanzados del modelado de caso de uso y luego terminamos nuestra explicación de caso de uso con algunas pistas y trucos.

Tratamos las relaciones que son posibles entre actores y actores y entre casos de uso y casos de uso. Estas relaciones son las siguientes:

- Generalización de actor: Una relación de generalización entre un actor más general y un actor más específico.
- Generalización de caso de uso: Una relación de generalización entre un caso de uso más general y un caso de uso más específico.
- `<<include>>`: Una relación entre casos de uso que permite que un caso de uso incluya comportamiento de otro.
- `<<extend>>`: Una relación entre casos de uso que permite que un caso de uso extienda su comportamiento con uno o más fragmentos de comportamiento de otro.

Es importante mantener todos los modelos lo más sencillo posible, por lo tanto utilice estas relaciones con discreción y solamente allí donde mejoren la claridad global del modelo de caso de uso. Es muy fácil emocionarse con `<<include>>` y `<<extend>>` en particular, pero debe evitarlo.

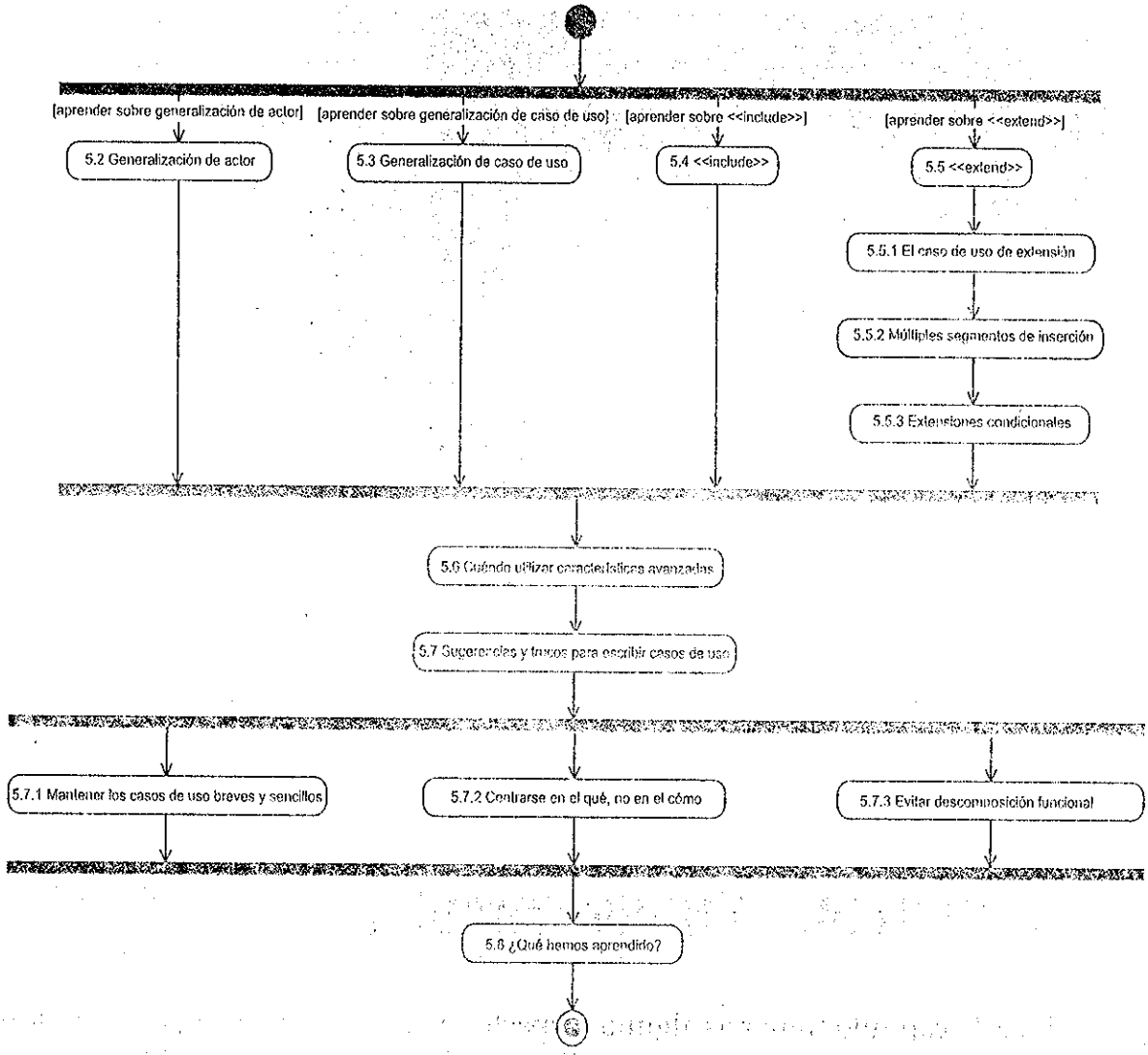


Figura 5.1.

5.2. Generalización de actor

En el ejemplo en la figura 5.2, puede ver muchas similitudes entre los dos actores, `Cliente` y `AgenteVentas`, en la forma que interactúan con el `Sistema ventas` (aquí, el `AgenteVentas` puede gestionar una venta en nombre del Cliente). Ambos actores activan los casos de uso `ListarProductos`, `ComprarProductos` y `AceptarPago`. De hecho, la única diferencia entre los dos actores es que `AgenteVentas` también activa el caso de uso `CalcularComisión`. Aparte del hecho de que esta semejanza en comportamiento proporciona muchas líneas cruzadas en el diagrama, parece indicar que existe cierto comportamiento común de actor que se podría resolver en un actor más generalizado.

Puede resolver este comportamiento común al utilizar la generalización de actor como se muestra en la figura 5.3. Cree un actor abstracto denominado `Comprador` que interactúa con los casos de uso `ListarProductos`, `ComprarProductos` y `AceptarPago`. `Cliente` y `AgenteVentas` se conocen como actores concretos

porque personas reales (u otros sistemas) podrían desempeñar esos roles. Sin embargo, `Comprador` es un actor abstracto ya que es una abstracción introducida simplemente para capturar el comportamiento común de los dos actores concretos, que ambos pueden actuar en un rol de compra.

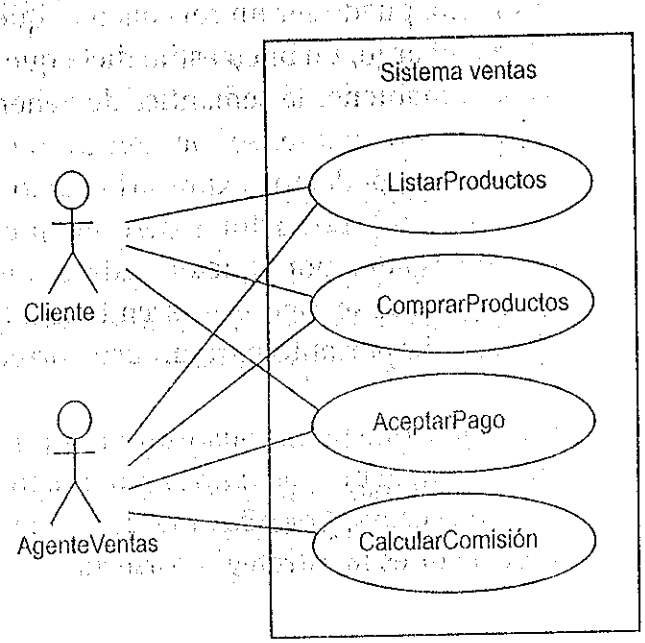


Figura 5.2.

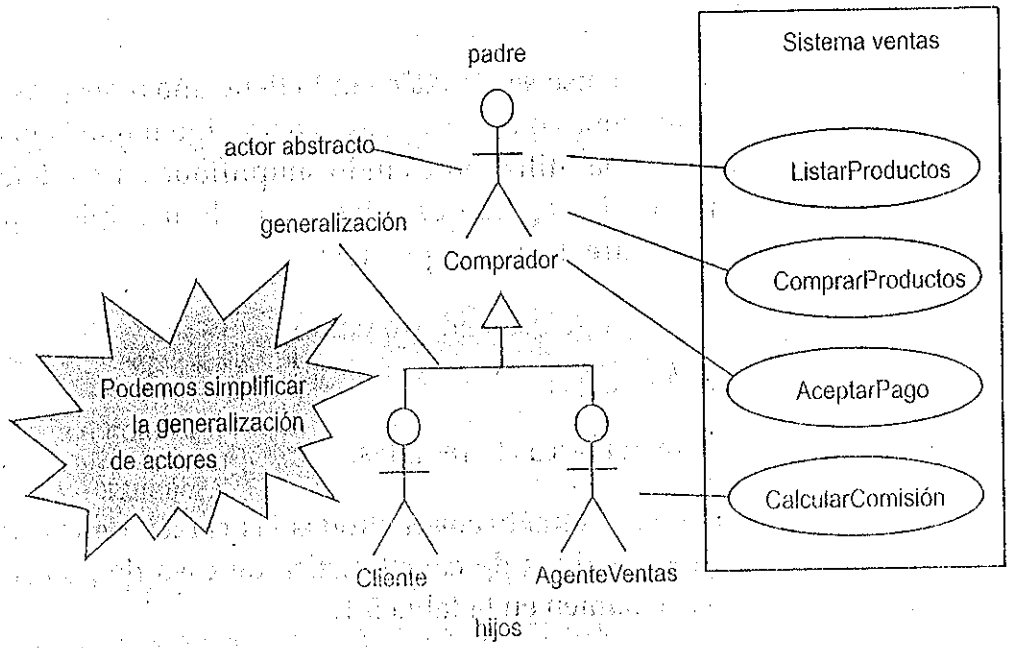


Figura 5.3.

`Cliente` y `AgenteVentas` heredan todos los roles y relaciones de los casos de uso de su padre abstracto. Por lo tanto, interpretando la figura 5.3, `Cliente` y `AgenteVentas` tienen interacciones con los casos de uso `ListarProductos`, `ComprarProductos` y `AceptarPago` que heredan de su padre, `Comprador`. Además, `AgenteVentas` tiene una interacción con el caso de uso `CalcularComisión` que no se hereda, es específica del actor `AgenteVentas`. Puede ver que un uso

juicioso de actores abstractos puede simplificar los diagramas de caso de uso. También simplifica la semántica de su modelo de caso de uso porque puede tratar elementos diferentes de la misma forma.

Merece la pena destacar que el actor padre en la generalización de actor no siempre tiene que ser abstracto, puede ser un rol concreto que una persona o sistema puede desempeñar. Sin embargo, un buen estilo dicta que los actores padre son normalmente abstractos para mantener la semántica de generalización sencilla.

Lo que hemos visto es que si dos actores se comunican con el mismo conjunto de casos de uso de la misma forma, podemos expresarlo como una generalización a otro actor (posiblemente abstracto). Los actores descendentes heredan los roles y relaciones a casos de uso albergados por el actor padre. Puede sustituir un actor descendente en cualquier lugar que se espere el ascendiente. Éste es el principio de sustitución que es una prueba importante para un uso correcto de generalización con cualquier clasificador.

En este ejemplo, es razonable que pueda sustituir un `AgenteVentas` o `Cliente` en cualquier lugar que se espere un `Comprador` (interactuando por ejemplo con los casos de uso `ListarProductos`, `ComprarProductos` y `AceptarPago`), por lo que la generalización de actor es la estrategia correcta.

5.3. Generalización de caso de uso

La generalización de caso de uso se utiliza cuando tiene uno o más casos de uso que son realmente especificaciones o un caso más general. Igual que la generalización de actor, solamente debería utilizarlo cuando simplifique su modelo de caso de uso. En la generalización de caso de uso, los casos de uso hijo representan formas más específicas del padre. Los hijos pueden:

- Heredar características de su caso de uso padre.
- Añadir nuevas características.
- Anular (cambiar) características heredadas.

El caso de uso hijo hereda automáticamente todas las características de su padre. Sin embargo, no todos los tipos de característica de caso de uso se pueden anular. Las restricciones se resumen en la tabla 5.1.

Tabla 5.1.

Característica de caso de uso	Heredar	Añadir	Anular
Relación.	S	S	N
Punto de ampliación.	S	S	N
Precondición.	S	S	N

Característica de caso de uso	Heredar	Añadir	Anular
Poscondición.	S	S	S
Paso en flujo principal.	S	S	S
Flujo alternativo.	S	S	S

En UML 1.5, los casos de uso también tienen atributos y operaciones, pero en UML 2 no. De hecho, los atributos y operaciones de caso de uso no parecen añadir ningún valor real y raramente se utilizaban y soportaban en herramientas UML. Según la especificación UML 1.5, las operaciones de un caso de uso ni siquiera se invocaban externamente, por lo que es difícil imaginar para qué estaban.

Por lo tanto, ¿cómo se documenta la generalización de caso de uso en especificaciones de caso de uso?

La especificación UML permanece en silencio en este punto pero existen varias técnicas estándar. Preferimos utilizar un lenguaje sencillo de etiquetas para resaltar las cinco posibilidades en un caso de uso hijo. Existen dos reglas para aplicar la técnica:

- Todo número del paso en el hijo está postfijado por el número de paso equivalente en el padre, si hay uno. Por ejemplo, `1.(2.) Algún paso.`
- Si el paso en el hijo anula un caso padre, se postfija por "o" y luego el número del paso en el padre. Por ejemplo, `6.(o6.) Otro paso.`

La tabla 5.2 resume la sintaxis para las cinco opciones en casos de uso hijo.

Tabla 5.2.

La característica se	Ejemplo de etiqueta
Hereda sin cambio.	`3.(3.)` El cliente incorpora los criterios requeridos.
Hereda y renumera.	`6.2(6.1)` El sistema dice al Cliente que no se pudieron encontrar productos coincidentes.
Hereda y anula.	`1.(o1.)` El Cliente selecciona "buscar libro".
Hereda, anula y renumera.	`5.2(o5.1)` El sistema muestra una página que indica los detalles de un máximo de cinco libros.
Añade.	`6.3` El sistema vuelve a mostrar la página de búsqueda "buscar libro".

La figura 5.4 muestra un extracto del diagrama de caso de uso de un sistema de ventas. Tenemos el caso de uso padre `BuscarProducto` y luego dos especializaciones de esto, `BuscarLibro` y `BuscarCD`.

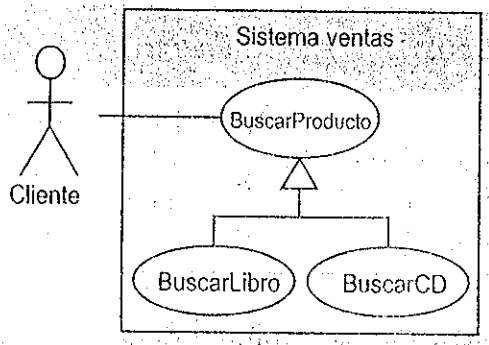


Figura 5.4.

La figura 5.5 muestra la especificación para el caso de uso padre `BuscarProducto`. Observe que se expresa en el nivel más alto de abstracción.

Caso de uso: BuscarProducto
ID: 6
Breve descripción: El Cliente busca un producto.
Actores principales: Cliente
Actores secundarios: Ninguno.
Precondiciones: Ninguna.
Flujo principal: 1. El Cliente selecciona "buscar producto". 2. El sistema pide al Cliente los criterios de búsqueda. 3. El Cliente escribe los criterios de búsqueda. 4. El sistema busca productos que coinciden con los criterios del Cliente. 5. Si el sistema encuentra productos que coinciden 5.1 El sistema muestra una lista de los productos que coinciden. 6. Sino 6.1 El sistema le dice al Cliente que no se han encontrado productos.
Postcondiciones: Ninguna.
Flujos alternativos: Ninguno.

Figura 5.5.

Uno de los casos de uso hijo, `BuscarLibro`, se muestra en la figura 5.6. Esto ilustra la aplicación de nuestro estándar para indicar anulación de nuevas características.

Como puede ver por la figura 5.6, el caso de uso hijo `BuscarLibro` es mucho más concreto. Especializa el padre más abstracto para tratar con un tipo específico de producto, libros.

Si el caso de uso padre no tiene flujo de eventos o un flujo de eventos que está incompleto, es un caso de uso abstracto. Los casos de uso abstractos son bastante comunes porque puede utilizarlos para capturar comportamiento en los niveles

más altos de abstracción. Puesto que los casos de uso abstracto tienen un flujo incompleto de eventos, pueden no ser ejecutados por el sistema. En lugar de un flujo de eventos, los casos de uso abstractos pueden tener un resumen de texto plano del comportamiento de alto nivel que sus hijos esperarán implementar. Puede situar esto en la breve descripción del caso de uso.

	Caso de uso: BuscarLibro
	ID: 7
	ID Padre: 6
	Breve descripción: El Cliente busca un libro.
	Actores principales: Cliente
	Actores secundarios: Ninguno.
	Precondiciones: Ninguna.
	Flujo principal: 1. (o1.) El Cliente seleccionar "buscar libro". 2. (o2.) El sistema pide al Cliente los criterios de búsqueda del libro como autor, título, ISBN o tema. 3. (3.) El Cliente escribe los criterios solicitados. 4. (o4.) El sistema busca libros que coinciden con los criterios del Cliente. 5. (o5.) Si el sistema encuentra libros que coinciden 5.1 El sistema muestra el de más ventas. 5.2 (o5.1) El sistema muestra los detalles de un máximo de cinco libros. 5.3 Para cada libro en la página, el sistema muestra el título, autor, precio e ISBN. 5.4 Mientras existan más libros, el sistema proporciona al Cliente la opción de mostrar la siguiente página de libros. 6. (6.) Sino 6.1 El sistema muestra el libro de mayor ventas. 6.2 (6.1) El sistema le dice al Cliente que no se han podido encontrar libros.
	Postcondiciones: Ninguna.
	Flujos alternativos: Ninguno.

anulado
anulado
heredado sin cambio
anulado
anulado
añadido
anulado y renumerado
añadido
añadido
heredado sin cambio
añadido
renumerado

Figura 5.6.

Como acaba de ver, es difícil mostrar características heredadas en casos de uso hijo. Tiene que utilizar algún tipo de lenguaje de etiqueta o convención tipográfica que los grupos de decisión normalmente encuentran confuso. Puesto que los casos de uso son todos comunicación con los grupos de decisión, ésta es una desventaja seria. Otra desventaja es que tiene que mantener manualmente coherencia entre los padres e hijos cuando uno de ellos cambia. Ésta es una tarea laboriosa y propensa a errores.

Un enfoque a este problema es restringir el caso de uso padre para que no tenga ningún flujo principal, sino solamente una breve descripción de su semántica. En ese caso, no tiene que preocuparse por la herencia o borrado. Este enfoque hace que la generalización de caso de uso sea sencilla y es una forma muy eficaz de mostrar

que uno o más casos de uso son simplemente variantes específicas de un caso de uso más general. El caso de uso más general le permite pensar en el sistema en la forma abstracta y puede incluir oportunidades para perfeccionar el sistema de software.

5.4. <<include>>

Escribir casos de uso puede ser muy repetitivo algunas veces. Suponga que está escribiendo un sistema de personal (véase la figura 5.7). Casi cualquier cosa que le pida al sistema que haga implicará primero localizar los detalles de un empleado específico. Si tuviera que escribir esta secuencia de eventos cada vez que necesitara los detalles del empleado, sus casos de uso se harían bastante repetitivos. La relación `<<include>>` entre los casos de uso le permite incluir el comportamiento de un caso de uso en el flujo de otro caso de uso.

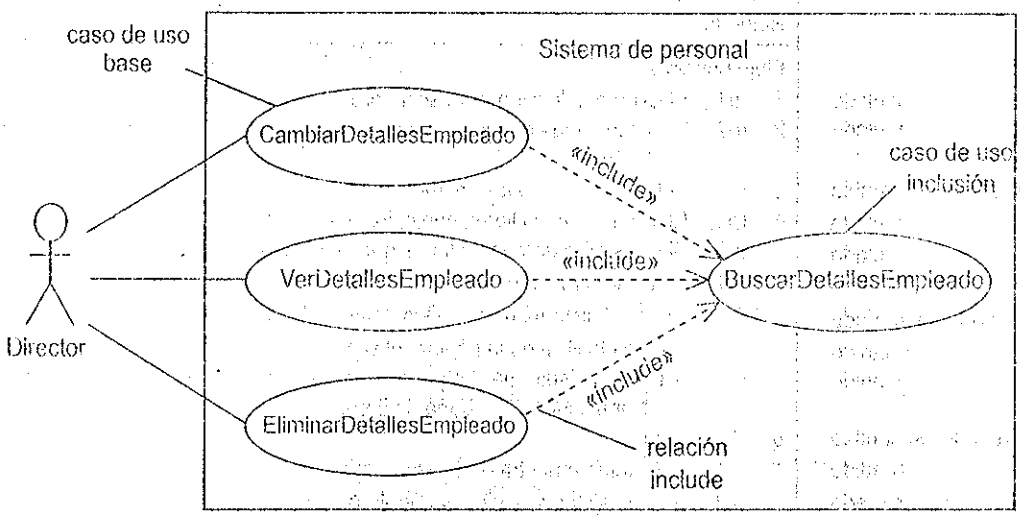


Figura 5.7.

Denominamos al caso de uso que incluye como el caso de uso base, y al caso de uso incluido como caso de uso de inclusión. El caso de uso de inclusión proporciona el comportamiento a su caso de uso base.

Debe especificar el punto exacto en el caso de uso base cuando necesita que se incluya el comportamiento del caso de uso de inclusión. La sintaxis para `<<include>>` es como una llamada de función, y de hecho tiene semántica similar.

La semántica de `<<include>>` es sencilla (véase la figura 5.8). El caso de uso base se ejecuta hasta que se alcanza el punto de inclusión, luego la ejecución se pasa al caso de uso de inclusión. Cuando termina el caso de uso de inclusión, el control regresa de nuevo al caso de uso base.

El caso de uso no está completo sin todos sus casos de uso de inclusión. Los casos de uso de inclusión forman partes integrales del caso de uso base. Sin embargo, los casos de uso de inclusión pueden o no estar completos. Si un caso de uso de inclusión no está completo, simplemente contiene un flujo parcial de eventos que solamente tendrán sentido cuando se incluya en una base apropiada.

Caso de uso: CambiarDetallesEmpleado
ID: 1
Breve descripción: El Director cambia detalles del empleado.
Actores principales: Director
Actores secundarios: Ninguno.
Precondiciones: 1. El Director se conecta al sistema.
Flujo principal: 1. include(BuscarDetallesEmpleado). 2. El sistema muestra los detalles del empleado. 3. El Director cambia los detalles del empleado. ...
Postcondiciones: 1. Se han cambiado los detalles del empleado.
Flujos alternativos: Ninguno.

Caso de uso: VerDetallesEmpleado
ID: 2
Breve descripción: El Director ve los detalles del empleado.
Actores principales: Director
Actores secundarios: Ninguno.
Precondiciones: 1. El Director se conecta al sistema.
Flujo principal: 1. include(BuscarDetallesEmpleado). 2. El sistema muestra los detalles del empleado. ...
Postcondiciones: 1. El sistema ha mostrado los detalles del empleado.
Flujos alternativos: Ninguno.

Caso de uso: EliminarDetallesEmpleado
ID: 3
Breve descripción: El Director elimina los detalles del empleado.
Actores principales: Director
Actores secundarios: Ninguno.
Precondiciones: 1. El Director se conecta al sistema.
Flujo principal: 1. include(BuscarDetallesEmpleado). 2. El sistema muestra los detalles del empleado. 3. El Director elimina los detalles del empleado. ...
Postcondiciones: 1. Los detalles del empleado se han eliminado.
Flujos alternativos: Ninguno.

Figura 5.8.

A menudo hacemos referencia a esto como un fragmento de comportamiento. En este caso, decimos que el caso de uso de inclusión no se puede instanciar, es decir, no se puede activar directamente por los actores, solamente se puede ejecutar cuando esté incluido en una base apropiada. Sin embargo, si los casos de uso de inclusión están completos, actúan como casos de uso normales y se pueden instanciar. Es bastante razonable activarlos por actores. Puede ver el caso de uso de inclusión, `BuscarDetallesEmpleado`, en la figura 5.9. Está incompleto y no se puede instanciar.

Caso de uso: BuscarDetallesEmpleado
ID: 4
Breve descripción: El Director busca los detalles del empleado.
Actores principales: Director.
Actores secundarios: Ninguno.
Precondiciones: 1. El Director se conecta al sistema.
Flujo principal: 1. El Director escribe el ID del empleado. 2. El sistema encuentra los detalles del empleado.
Postcondiciones: 1. El sistema ha encontrado los detalles del empleado.
Flujos alternativos: Ninguno.

Figura 5.9.

5.5. <<extend>>

`<<extend>>` proporciona un medio para insertar un nuevo comportamiento en un caso de uso existente (véase la figura 5.10). El caso de uso base proporciona un conjunto de puntos de extensión que son enganches donde se puede añadir nuevo comportamiento, el caso de uso de extensión proporciona un conjunto de segmentos de inserción que se pueden insertar en el caso de uso base en esos enganches. Como verá en breve, la relación `<<extend>>` se puede utilizar para especificar exactamente qué puntos de extensión en el caso de uso base se están extendiendo.

Lo que es interesante de `<<extend>>` es que el caso uso base no sabe nada de los casos de uso de extensión; simplemente proporciona enganches para estos. De hecho, el caso de uso base está perfectamente completo sin sus extensiones. Esto es muy diferente de `<<include>>`, donde los casos de uso base estaban incompletos sin sus casos de uso de inclusión. Además, los puntos de extensión no están realmente insertados en el flujo de eventos del caso de uso base, sino que están añadidos a una capa en la parte superior del flujo de eventos.

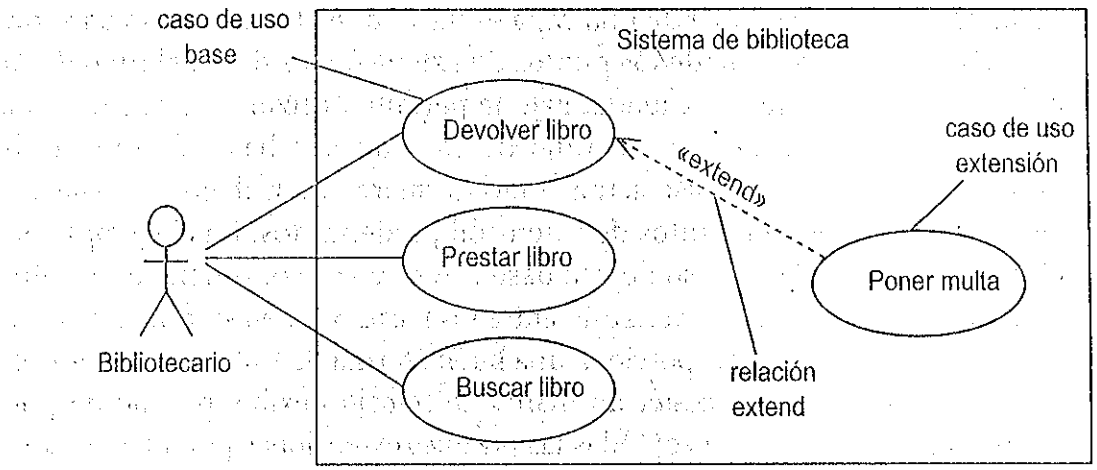
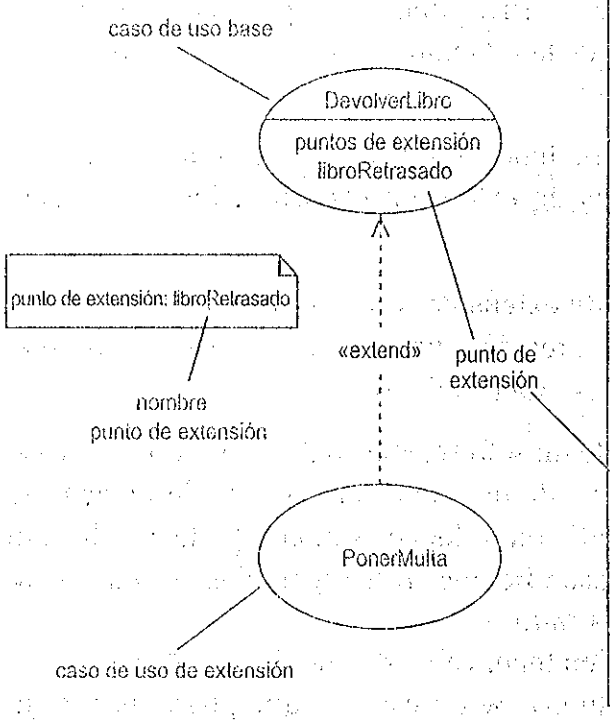


Figura 5.10.

Los puntos de extensión están indicados en el flujo de eventos del caso de uso base como se muestra en la figura 5.11. También puede mostrar puntos de extensión en el diagrama de caso de uso al listarlos en un nuevo compartimiento en el icono de caso de uso base.



Caso de uso: DevolverLibro
ID: 9
Breve descripción: El Bibliotecario devuelve un libro prestado.
Actores principales: Bibliotecario
Actores secundarios: Ninguno.
Precondiciones: 1. El Bibliotecario se conecta al sistema.
Flujo principal: 1. El Bibliotecario escribe el número del prestatario. 2. El sistema muestra los detalles del prestatario incluida la lista de libros prestados. 3. El Biblioteca busca el libro a devolver en la lista de libros. punto de extensión: libroRetrasado 4. El Bibliotecario devuelve el libro. ...
Postcondiciones: 1. El libro se ha devuelto.
Flujos alternativos: Ninguno.

Figura 5.11.

Observe que los puntos de extensión en el flujo principal no están numerados. En su lugar, aparecen entre los pasos numerados del flujo. De hecho, UML indica explícitamente que los puntos de extensión existen solamente en una capa sobre el flujo principal. Por lo tanto no son parte del flujo principal. Puede pensar en esta capa como un acetato sobre el flujo principal donde se graban los puntos de exten-

sión. Lo interesante de esta idea de una capa es hacer que el caso de uso base fluya completamente independiente de los puntos de extensión. Es decir, el flujo de caso de uso base no sabe dónde se extiende. Esto le permite utilizar <<extend>> para realizar extensiones arbitrarias a un flujo de caso de uso base. Cuando utiliza <<extend>>, el caso de uso base actúa como un marco de trabajo modular en el que conectar extensiones en puntos de extensión predefinidos. En el ejemplo en la figura 5.11, puede ver que el caso de uso base DevolverLibro tiene un punto de extensión denominado libroRetrasado entre los pasos 3 y 4 en su flujo de eventos. Puede ver que <<extend>> proporciona una buena forma de tratar con casos excepcionales o casos en los que necesita un marco de trabajo flexible porque no puede predecir (o simplemente no conoce) todas las posibles extensiones por adelantado.

5.5.1. El caso de uso de extensión

Los casos de uso de extensión son por lo general casos de uso no completos y por lo tanto no se pueden instanciar. Simplemente constan de uno o más fragmentos de comportamiento conocidos como segmentos de inserción. La relación <<extend>> especifica el punto de extensión en el caso de uso base donde se insertará el segmento de inserción. Se aplican las siguientes reglas:

- La relación <<extend>> debe especificar uno o más puntos de extensión en el caso de uso base o se asume que la relación <<extend>> hace referencia a todos los puntos de extensión.
- El caso de uso de extensión debe tener el mismo número de segmentos de inserción ya que existen puntos de extensión especificados en la relación <<extend>>.
- Es legal para dos casos de uso de extensión <<extend>> el mismo caso de uso base en el mismo punto de extensión. Pero si sucede esto, el orden en el que se ejecutan las extensiones es indeterminado.

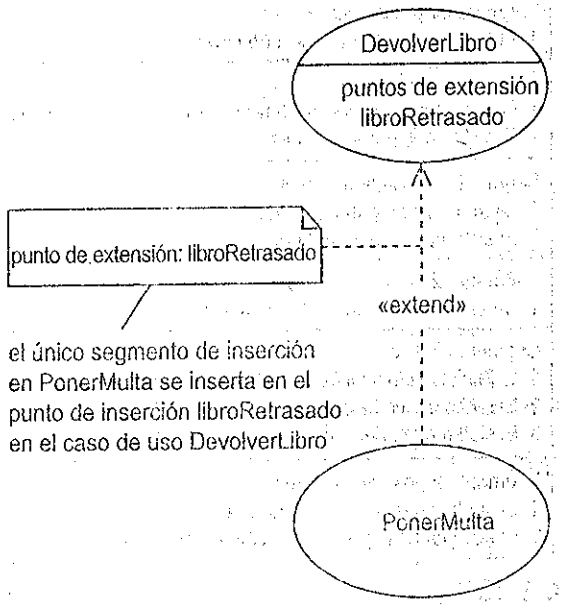
En el ejemplo en la figura 5.12, existe un solo segmento de inserción en el caso de uso de extensión PonerMulta. El caso de uso de extensión puede tener precondiciones o poscondiciones. Las precondiciones deben estar completas; de lo contrario, el segmento no se ejecuta. Las poscondiciones restringen el estado del sistema después de que el segmento se ha ejecutado.

Los casos de uso de extensión pueden tener casos de uso de extensión o casos de uso de inclusión. Sin embargo, tendemos a evitar esto, ya que puede hacer que el modelo de caso de uso sea demasiado complejo.

5.5.2. Múltiples segmentos de inserción

Puede tener múltiples segmentos de inserción en un caso de uso de extensión. Esto es de utilidad cuando no puede capturar la extensión en un solo segmento porque necesita regresar al flujo principal del caso de uso base para hacer algo. En

el ejemplo en la figura 5.13, puede imaginar que después de grabar e imprimir una multa, regresamos al flujo principal para procesar cualquier otro libro pasado de plazo y, por último, en el punto de extensión `pagarMulta`, damos a la persona la opción de pagar la multa total. Esto es mucho más eficaz que tener que aceptar el pago para cada multa individualmente, que habría sido el caso si hubiéramos combinado los dos segmentos en `PonerMulta`.



Caso de uso de extensión: PonerMulta
ID: 10
Breve descripción: Segmento 1: El Biblioteca graba e imprime una multa.
Actores principales: Bibliotecario.
Actores secundarios: Ninguno.
Segmento 1 precondiciones: 1. El libro devuelto está fuera de plazo.
Segmento 1 flujo: 1. El Bibliotecario escribe los detalles de la multa en el sistema. 2. El sistema imprime la multa.
Segmento 1 postcondiciones: 1. La multa se ha grabado en el sistema. 2. El sistema ha impreso la multa.

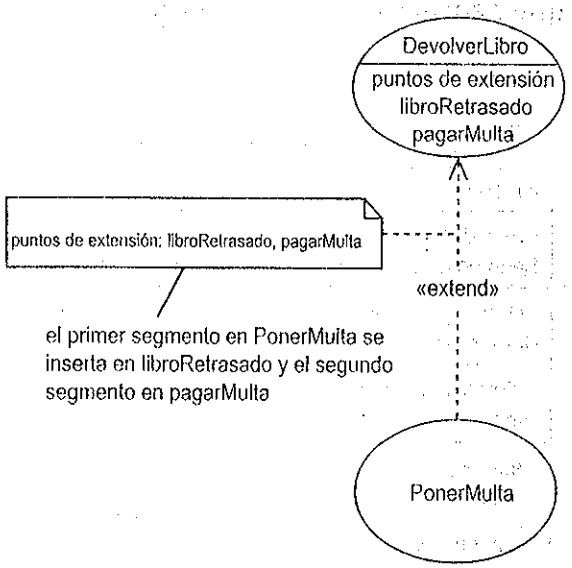
Figura 5.12.

Cuando crea casos de uso de extensión con múltiples segmentos, tiene que etiquetar cada segmento con un número según se muestra en la figura 5.13. Esto es porque el orden de los segmentos es importante; el primer segmento se inserta en el primer punto de extensión, etc. Por lo tanto, debe tener cuidado de escribir los segmentos en el orden correcto, y conservar este orden.

5.5.3. Extensiones condicionales

El ejemplo en la figura 5.14 muestra un sistema de biblioteca algo más bueno en el que a los prestatarios se les da un aviso la primera vez que el libro se devuelve pasado el plazo y se les penaliza solamente después. Podemos modelar esto al añadir un nuevo caso de uso de extensión, `EmitirAviso`, y luego situar condiciones en las relaciones `<<extend>>`. Las condiciones sobre expresiones booleanas, y la inserción se realiza si, y sólo si, la expresión evalúa como verdadero.

Observe que el caso de uso de extensión `EmitirAviso` solamente se extiende en el punto de extensión `libroRetrasado`. Sin embargo (como antes), el caso de uso de extensión `PonerMulta` se extiende tanto en `libroRetrasado` y `pagarMulta`. Esto le dice inmediatamente que `EmitirAviso` (véase la figura 5.15) contiene exactamente un segmento de inserción, mientras que `PonerMulta` (como ya hemos visto) contiene dos.



Caso de uso de extensión: PonerMulta
ID: 10
Breve descripción: Segmento 1: El Bibliotecario graba e imprime una multa. Segmento 2: El Bibliotecario acepta pagar por la multa.
Actores principales: Bibliotecario.
Actores secundarios: Ninguno.
Segmento 1 precondiciones: 1. El libro devuelto está fuera de plazo.
Segmento 1 flujo: 1. El Bibliotecario escribe los detalles de la multa en el sistema. 2. El sistema imprime la multa.
Segmento 1 postcondiciones: 1. La multa se ha grabado en el sistema. 2. El sistema ha impreso la multa.
Segmento 2 predondiciones: 1. Se entrega una multa al prestatario.
Segmento 2 flujo: 1. El Bibliotecario acepta pago de la multa del prestatario. 2. El Bibliotecario incorpora la multa pagada en el sistema. 3. El sistema imprime un recibo de la multa pagada.
Segmento 2 postcondiciones: 1. La multa se graba como pagada. 2. El sistema ha impreso un recibo de la multa.

Figura 5.13.

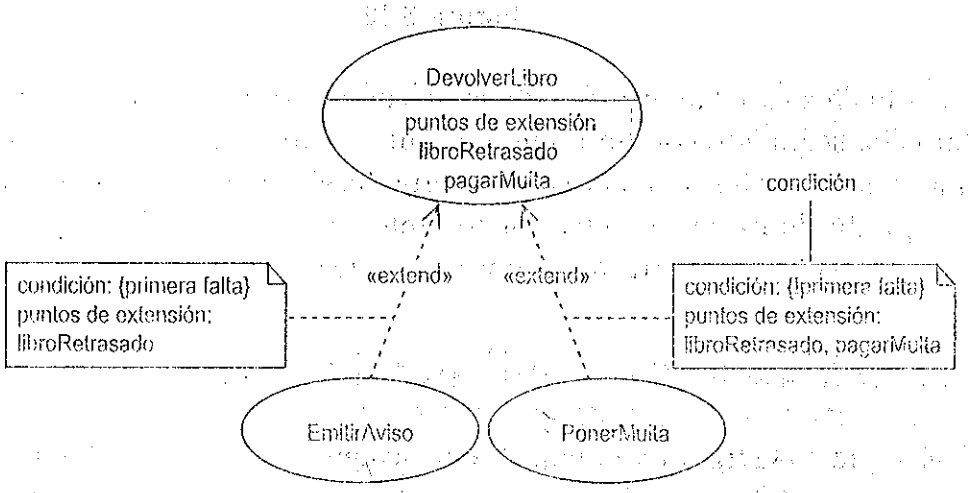


Figura 5.14.

5.6. Cuándo utilizar características avanzadas

Utilice características avanzadas cuando simplifiquen su modelo de caso de uso. Hemos encontrado que los mejores modelos de caso de uso son sencillos. Recuerde que el modelo de caso de uso es una declaración de requisitos y, como tal, debe estar accesible para los grupos de decisión así como para los modeladores. Un sencillo modelo de caso de uso que solamente utiliza características avanzadas

escasamente, si alguna, es preferible a uno que usa demasiadas características avanzadas, incluso si ese modelo es en cierta forma más elegante desde la perspectiva de un modelador.

Caso de uso de extensión: EmitirAviso
ID: 11
Breve descripción: Segmento 1: El Bibliotecario envía un aviso.
Actores principales: Bibliotecario.
Actores secundarios: Ninguno.
Segmento 1 precondiciones: 1. El libro devuelto está fuera de plazo.
Segmento 1 flujo: 1. El Biblioteca incorpora los detalles del aviso en el sistema.
Segmento 1 postcondiciones: 1. El aviso se ha grabado en el sistema.

Figura 5.15.

Basándose en nuestra experiencia del modelado de caso de uso, en muchas empresas diferentes hemos encontrado que:

- Por lo general, los grupos de decisión pueden entender fácilmente los actores y casos de uso con poca formación.
- Los grupos de decisión encuentran la generalización de actor más difícil de entender.
- Un gran uso de `<<include>>` puede hacer que los modelos de caso de uso sean más difíciles de entender; los grupos de decisión y los modeladores tienen que ver más de un caso de uso para hacerse una idea completa.
- Los grupos de decisión tienen mucha dificultad con `<<extend>>`; esto puede ser cierto incluso con explicaciones muy detalladas.
- Un sorprendente número de modeladores de objeto no entiende bien la semántica de `<<extend>>`.
- La generalización de caso de uso se debería evitar a menos que se utilicen casos de uso padre abstractos (en lugar de concretos), de lo contrario añade demasiada complejidad a los casos de uso hijos.

5.7. Sugerencias para escribir casos de uso

En este apartado proporcionamos algunas sugerencias y trucos para escribir casos de uso.

5.7.1. Mantener los casos de uso breves y sencillos

Nuestro eslogan para los casos de uso es "mantenlos breves, mantenlos sencillos". Incluye solamente el detalle suficiente para capturar los requisitos. Desafortunadamente, algunos proyectos tienen miedo de los que es breve y sencillo y están enamorados de grandes cantidades de documentación. Grady Booch llamaba a esta tendencia "envidiar el papel".

Una buena regla general a seguir es asegurarse de que el flujo principal de un caso de uso cabe en una sola hoja de papel. Más de éstas y el caso de uso es demasiado largo. De hecho, encontrará que muchos casos de uso ocupan menos de media página.

Empiece por simplificar el texto (recuerde utilizar frases declarativas cortas). Elimine cualquier detalle de diseño. Si sigue siendo demasiado largo, vuelva a analizar el problema. ¿Es posible que exista más de un caso de uso? ¿Puede resolver flujos alternativos?

5.7.2. Centrarse en el qué, no en el cómo

Recuerde que está escribiendo casos de uso para solucionar lo que los actores necesitan que haga el sistema, no cómo el sistema debería hacerlo. El cómo viene después en el workflow de diseño. Confundir el qué con el cómo es un problema que vemos continuamente. El escritor del caso de uso inventa alguna solución cuando escribe el caso de uso. Por ejemplo, considere el fragmento de caso de uso que se proporciona a continuación.

```
...
4. El sistema pide al Cliente que confirme el pedido.
5. El Cliente pulsa el botón Aceptar.
...
```

En este paso, el escritor del caso de uso ha imaginado algún tipo de interfaz de usuario: un formulario con un botón Aceptar. Debido a esto, el caso de uso ha dejado de ser una declaración de requisitos, es un diseño primitivo. Una mejor forma de expresar el paso 5 es la siguiente:

```
...
5. El Cliente acepta el pedido.
...
```

Mantenga los detalles del diseño (que todavía no conoce) fuera del caso de uso.

5.7.3. Evite descomposición funcional

Un error común en el análisis de caso de uso es crear un conjunto de casos de uso de "alto nivel" y luego desglosarlos en un conjunto de casos de uso de bajo nivel, y así sucesivamente hasta que acabe con casos de uso "primitivos" que están suficien-

temente detallados para implantarse. Este enfoque al diseño de software se conoce como descomposición funcional y es erróneo cuando se aplica al modelado de casos de uso.

Examinemos un ejemplo. En la figura 5.16, el analista ha definido la operación de un sistema de biblioteca utilizando un sencillo caso de uso de alto nivel, `GestionarBiblioteca`, y luego lo ha descompuesto en casos de uso cada vez con más niveles de detalle, creando una descomposición funcional.

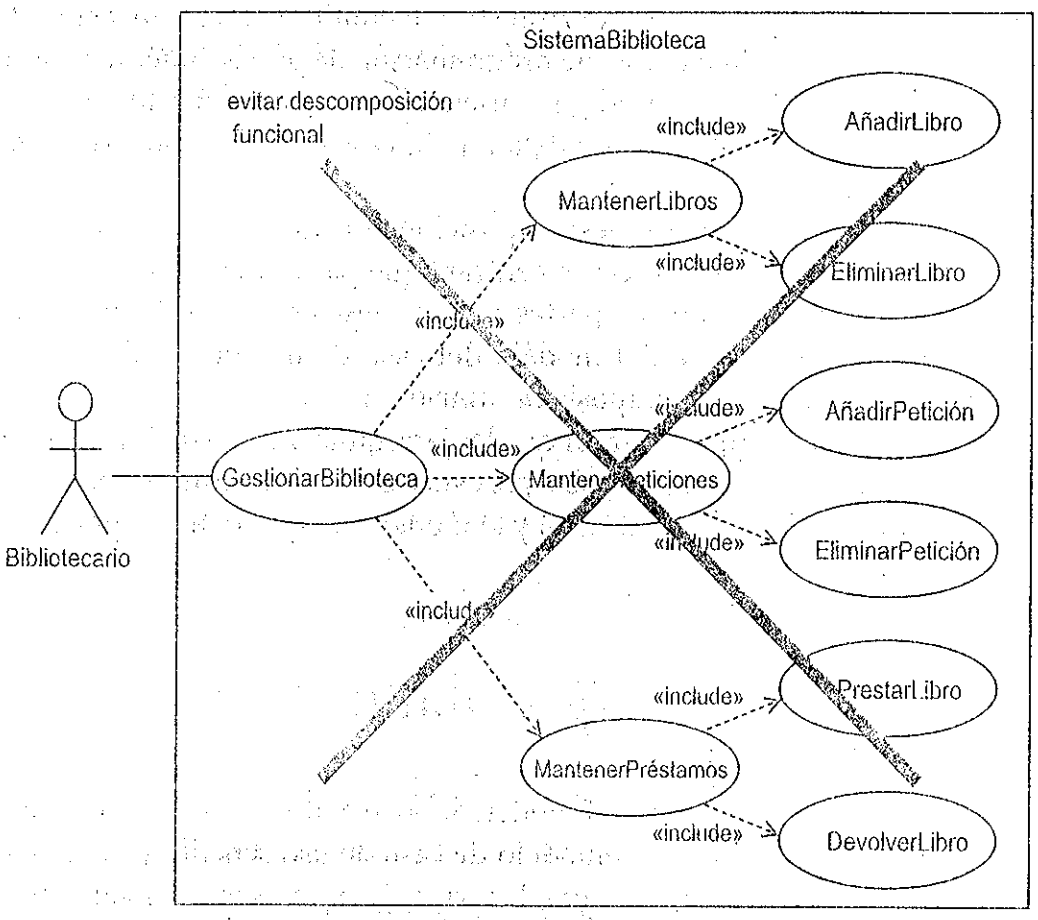


Figura 5.16.

Muchos analistas no orientados a objetos encontrarán la figura 5.16 bastante posible. Sin embargo, como modelo de caso de uso hay muchos errores en ella.

- En lugar de centrarse en capturar los requisitos, el modelo se centra en estructurar esos requisitos de forma artificial; existen muchas descomposiciones posibles.
- El modelo describe el sistema como un conjunto de funciones anidadas. Sin embargo, los sistemas orientados a objetos son conjuntos de objetos que cooperan enviando mensajes de un lado a otro. Aquí hay un mal emparejamiento conceptual.
- Solamente el nivel más bajo de casos de uso, `AñadirLibro`, `EliminarLibro`, `AñadirPetición`, `EliminarPetición`, `PrestarLibro` y `DevolverLibro`,

tiene especificaciones interesantes. Los niveles más altos invocan a los más bajos y no añaden nada al modelo en términos de capturar requisitos.

- El modelo es difícil de entender para los grupos de decisión. Existen varios casos de uso abstractos (`GestionarBiblioteca`, `MantenerLibros`, `MantenerPeticiones` y `MantenerPréstamos`) y numerosas relaciones `<<include>>` a los niveles más bajos.

El uso de un enfoque de descomposición funcional sugiere que un analista ha pensado en el sistema de forma errónea. A menudo es una indicación de que el analista está formado en técnicas de programación de procedimientos más tradicionales y no ha entendido el paradigma orientado a objetos. En este caso, siempre es una buena idea emplear a un modelador de casos de uso con experiencia para proporcionar consejos y revisiones.

No todos los ejemplos de descomposición funcional son tan obvios como el ejemplo en la figura 5.16. A veces encontrará que partes del modelo de caso de uso son adecuadas, mientras otras partes se han descompuesto. Es una buena idea comprobar cualquier parte del modelo del caso de uso que tenga una jerarquía profunda para posible descomposición funcional.

Por último, deberíamos destacar que las jerarquías surgen de forma natural en el modelado de casos de uso. Sin embargo, estas jerarquías naturales no tienen nunca más de un nivel (o dos como mucho) y el modelo completo nunca parte de un caso de uso único.

5.8. ¿Qué hemos aprendido?

Ha aprendido técnicas para el modelado avanzado de caso de uso. Su objetivo siempre debería ser generar un modelo de caso de uso sencillo y fácil de entender que capture la información necesaria de forma clara y concisa. Personalmente, siempre preferimos utilizar un modelo de caso de uso que no utilice ninguna de las características avanzadas en lugar de uno en el que exista demasiada generalización, `<<include>>`, y `<<extend>>` que sea difícil saber qué es lo que sucede. Como regla general, "si tiene dudas, no lo incluya". Ha aprendido lo siguiente.

- La generalización de actor le permite destacar comportamientos de actor padre que son comunes a dos o más actores.
  - El actor padre está más generalizado que sus hijos, y los hijos están más especializados que su padre.
  - Puede sustituir un actor hijo en cualquier lugar que se espere un actor padre, éste es el principio de sustitución.
  - El actor padre a menudo es abstracto; especifica un rol abstracto.
  - Los actores hijos son concretos; especifican roles concretos.

- La generalización de actor puede simplificar los diagramas de caso de uso.

- La generalización de caso de uso le permite destacar características que son comunes a dos o más casos de uso en un caso de uso padre.
  - Los casos de uso hijo heredan todas las características de su caso de uso padre.
  - Los casos de uso hijo pueden añadir nuevas características.
  - Los casos de uso hijo pueden anular características padre, excepto relaciones y puntos de extensión.
  - Utilizamos una sencilla convención de etiqueta en los casos de uso hijo:
    - Heredada sin cambio, `3.(3.)`
    - Heredada y renumerada, `6.2(6.1)`
    - Heredada y anulada `1.(o1.)`
    - Heredada, anulada y renumerada `5.2 (o5.1)`
    - Añadida, `6.3`
  - Un buen estilo indica que el caso de uso padre debería ser abstracto.

- `<<include>>` le permite resolver pasos repetidos en varios flujos de caso de uso en un caso de uso aparte que incluye donde sea necesario.
  - `include(NombreCasoUso)` se utiliza para incluir el comportamiento de otro caso de uso.
  - El caso de uso que se incluye es el caso de uso base.
  - El caso de uso incluido es el caso de uso de inclusión.
  - El caso de uso base no está completo sin todos sus casos de uso de inclusión.
  - Los casos de uso de inclusión pueden ser:
    - **Completos:** Son casos de uso normales y se pueden instanciar.
    - **Incompletos:** Contienen solamente un fragmento de comportamiento y no se pueden instanciar.

- `<<extend>>` añade nuevo comportamiento a un caso de uso base existente.
  - El caso de uso base tiene puntos de extensión en una capa en su flujo de eventos. Los puntos de extensión ocurren entre los pasos en el flujo de eventos.
  - Los casos de uso de extensión proporcionan segmentos de inserción. Éstos son fragmentos de comportamiento que se pueden "conectar" a los puntos de extensión.

- La relación <<extend>> entre los casos de uso de extensión y el caso de uso base especifica los puntos de extensión en los cuales se conectan los segmentos de inserción del caso de uso de extensión.
- El caso de uso base está completo sin los segmentos de inserción; no sabe nada sobre posibles segmentos de inserción, simplemente proporciona enganches para ellos.
- El caso de uso de extensión está por lo general no completo; normalmente sólo consta de uno o más segmentos de inserción; puede ser también un caso de uso completo, aunque es raro.
- Si el caso de uso de extensión tiene precondiciones, se deben completar; de lo contrario, el caso de uso de extensión no se ejecuta.
- Si el caso de uso de extensión tiene poscondiciones, éstas restringen el estado del sistema después de ejecutarse el caso de uso de extensión.
- Un caso de uso de extensión puede contener muchos segmentos de inserción.
- Dos o más casos de uso de extensión pueden extender el mismo caso de uso base en el mismo punto de extensión; el orden de ejecución de cada caso de uso de extensión es indeterminado.
- Las extensiones condicionales, condiciones de protección booleanas en la relación <<extend>> permiten una inserción si es verdadero, e impiden una inserción si es falso.

- Utilice características avanzadas de la siguiente manera:
  - Generalización de actor: Sólo cuando simplifique el modelo.
  - Generalización de caso de uso: Considere no utilizarlo o solamente utilizarlo con padres abstractos.
  - <<include>>: Utilícelo solamente si simplifica el modelo; tenga cuidado con un uso excesivo, ya que esto hace que el modelo de caso de uso se convierta en una descomposición funcional.
  - <<extend>>: Considere no utilizarlo, pero si lo hace, tenga cuidado de asegurarse de que todos los modeladores y grupos de decisión presentados en el modelo entienden y están de acuerdo en su semántica.
- Sugerencias y trucos para escribir casos de uso:
  - Mantenga los casos de uso breves y sencillos.
  - Céntrese en el qué, no en el cómo.
  - Evite descomposición funcional.