Casos a incluir y casos a extender

Un tema que genera mucha polémica entre la gente que modela casos de uso es la elección entre la relación de <<include>> y <<extend>>. Lo peor es que muchas de esas discusiones generan muy poco valor en el resultado final en el modelo y en cambio quitan tiempo valioso del proyecto. Esto se debe a que dichas relaciones, muchas veces no son del todo comprendidas por la persona que la modela, y mucho menos son comprendidas por las personas que leen el modelo. Así que al final no se le saca el provecho que en todo caso debería de tener dicha elección.

Es mejor mantener el modelo simple, incluso si esto significa utilizar menos elementos gráficos de UML, si eso va a facilitar el modelado y la comunicación en el proyecto. Pero, después de todo este tiempo y de los diferentes temas que hemos venido tratando, es importante avanzar y adentrarnos en algunos pormenores del lenguaje unificado.

Antes que todo, ¿qué es el "include" y el "extend"?

Gráficamente lo que mostramos es una relación de dependencia entre el par de casos de uso, con el nombre correspondiente al tipo de relación de la que se trate: ya sea <<include>> o <<extend>>.

Estas, son relaciones que usamos para ligar gráficamente dos casos de uso, cuyos flujos de eventos están unidos, normalmente en una sola sesión del usuario. En otras palabras, un caso de uso que está ligado, por medio de una de estas dos relaciones, a otro caso de uso; también podría leerse o ejecutarse como un sólo caso de uso en lugar de dos. Obviamente, hay una razón por la cual decidimos separarlos en dos, lo cual explicaremos más adelante.

Imagina el conjunto de pasos que ocurren al realizar una compra en una tienda departamental. Seguramente no tendrás problema en visualizar el conjunto de pasos para solicitar la autorización de la tarjeta de crédito con la que vas a pagar, ligado a los pasos donde el vendedor registra los productos a comprar. Es decir, los flujos de

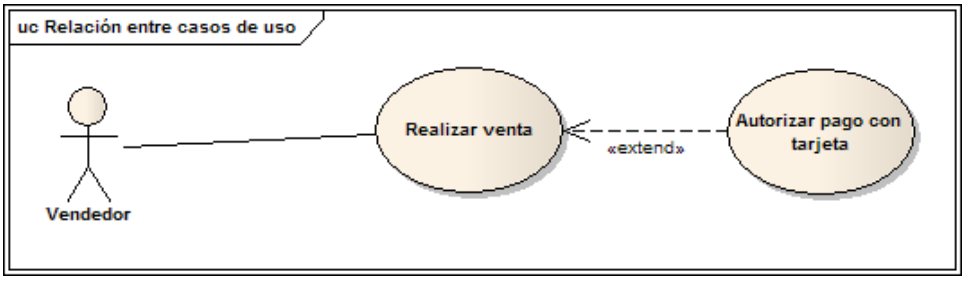

Figura 1. Relacionando casos de uso

Include. En términos muy simples, cuando relacionamos dos casos de uso con un "include", estamos diciendo que el primero (el caso de uso base) incluye al segundo (el caso de uso incluido). Es decir, el segundo es parte esencial del primero. Sin el segundo, el primero no podría funcionar bien; pues no podría cumplir su objetivo. Para una venta en caja, la venta no puede considerarse completa si no se realiza el proceso para cobrarla en ese momento. El caso de uso "Cobrar Renta" está incluido en el caso de uso "Rentar Video", o lo que es lo mismo "Rentar Video" incluye (<<include>>) "Cobrar Renta".

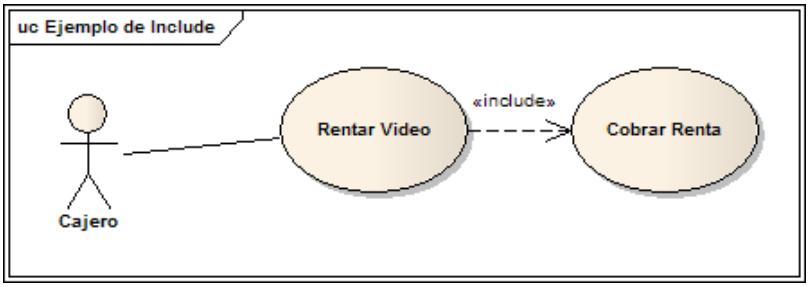

Figura 2. Ejemplo de Include

Extend. La polémica al querer seleccionar una de las dos relaciones es que en el "extend" también podemos ver, desde la perspectiva del usuario, a los dos flujos como si fueran uno sólo. Y en ciertos escenarios el caso de uso base no podría cumplir su objetivo si no se ejecutara la extensión. Pero, una de las diferencias básicas es que en el caso del "extend" hay situaciones en que el caso de uso de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del caso de uso base. En cambio en el "include" es necesario que ocurra el caso incluído, tan sólo para satisfacer el objetivo del caso de uso base. Ejemplo: Puedes "Realizar Venta" sin "Acumular Puntos de Cliente VIP", cuando no eres un cliente VIP. Pero, si eres un cliente VIP sí acumularás puntos. Por lo tanto, "Acumular Puntos" es una extensión de "Realizar Venta" y sólo se ejecuta para cierto tipo de ventas, no para todas.

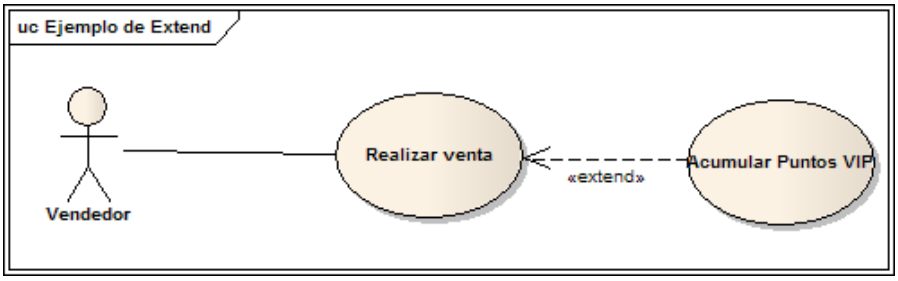

Figura 3. Ejemplo de Extend

Uno de los riesgos que existe cuando la gente sabe que tiene estas relaciones como un elemento a utilizar en sus modelos de caso de uso, consiste en su abuso. Mucha gente, y sobre todo la que arrastra prácticas de métodos estructurados, la suele utilizar en exceso. No es raro ver modelos de casos de uso que llegan a tener decenas de inclusiones y extensiones, incluso las inclusiones y extensiones se vuelven a extender a varios niveles, generando una maraña de casos de uso que no ofrecen valor al ser mostrados explícitamente.

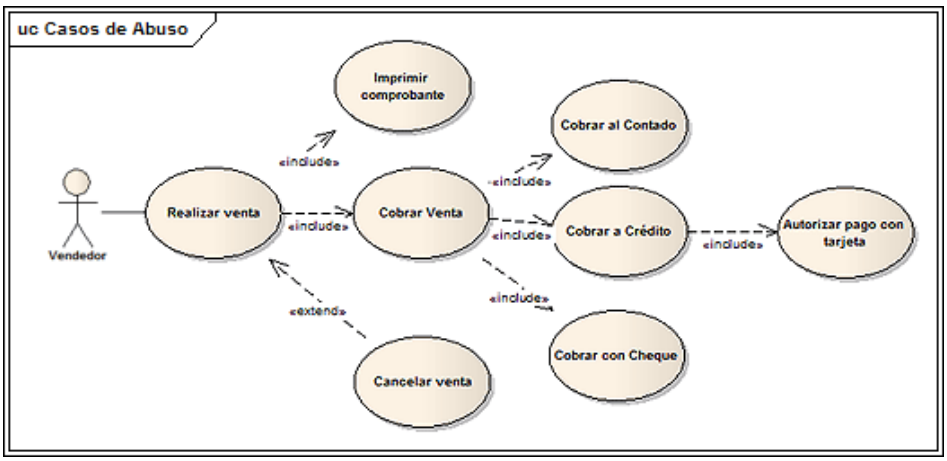

Figura 4. Abuso de relaciones

Es importante comprender que el objetivo de estos tipos de relaciones NO consiste (remarco la negación) en motivar la división de los casos de uso en la mayor cantidad de pedazos. Debe de existir una razón importante para que decidamos dividir un caso de uso en dos que serán unidos por medio de alguna de estas relaciones. Si entendemos esto y somos congruentes, obtendremos un beneficio real para el proyecto; fin último del uso de UML.

La razón por la que la gente suele partir sus caso de uso en infinidad de “include” y “extend” es porque quieren conocer, entender y comunicar el máximo detalle de los casos de uso en el diagrama. Hay quien llega a utilizar, erróneamente, estas relaciones para mostrar el orden en que se ejecutan estos casos de uso. Debemos de recordar que al modelar el diagrama de casos de uso no buscamos analizar el detalle, y mucho menos los flujos. Todo ese detalle lo podremos plasmar en otro tipo de diagramas, como los diagramas de interacción, de actividad, de estados, o simplemente un texto en una especificación.

Relaciones de Análisis o Diseño

Otra situación donde abusamos de estas relaciones se da cuando queremos representar la unión de casos de uso por una decisión de diseño del sistema, específicamente por una decisión de navegabilidad entre funciones. Pensemos en cierta funcionalidad en un sistema, la cual corresponde a la ejecución de cierto caso de uso (por ejemplo “Registrar Préstamo de un Video”). Y estando en dicho caso de uso tienes a la vista en la pantalla, y decides utilizar, un botón que te permitiría iniciar otro caso de uso que tiene poco o nada que ver con el objetivo del caso de uso inicial (digamos, “Consultar Promociones”). Esto no debería de mostrarse como una relación entre estos dos casos de uso en el modelo.

No deberíamos modelar al primer caso de uso incluyendo ni siendo extendido con el segundo caso de uso ni viceversa, pues la razón por la que se ligaron (no

de uso normalmente tiene que ver poco con que exista o no una relación entre dichos casos de uso.

Reuso: Evitando el Retrabajo

Una de las razones por las cuales deberías de considerar el uso de este tipo de relaciones, es porque identificas que hay pasos que son iguales en dos o más casos de uso. No querrás tener que escribir y darle mantenimiento a esos pasos en los documentos asociados a cada uno de ellos. Peor aún, no querrás correr el riesgo de que esos pasos se diseñen, programen y prueben de maneras diferentes y con esfuerzos aislados por ti o tu equipo de desarrollo. Finalmente son la misma cosa, ¿para qué querríamos trabajar doble? Lo que queremos es economizar, ser más eficientes en el desarrollo, y ahí es cuando viene el beneficio de identificar estos tipos de relaciones; porque es una oportunidad de identificar reuso.

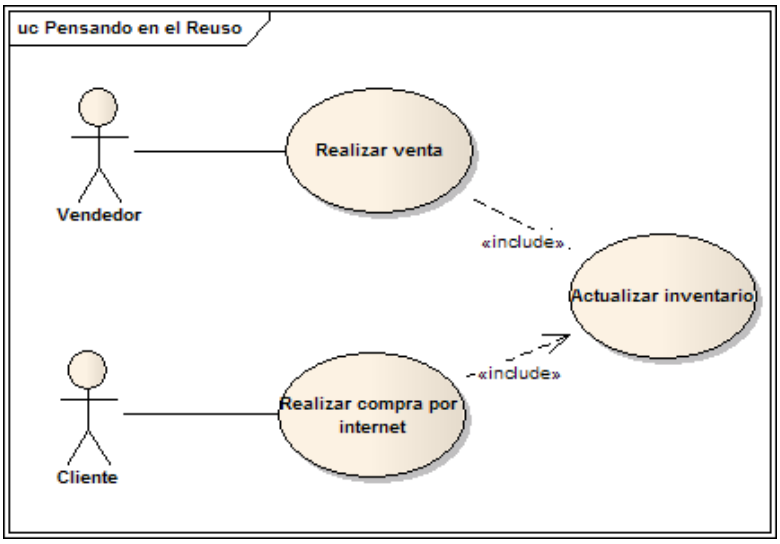

Figura 5. Identificación de reuso

Si te sientes preparado para desarrollar modelos de casos de uso más sofisticados y de mayor valor, entonces considera la posibilidad de utilizar estos tipos de relaciones. Sólo asegúrate de aprovecharlas adecuadamente, buscando el beneficio real que deberían de proporcionar en tu modelo y proyecto con base en las recomendaciones mencionadas. Y recuerda unificar los criterios dentro de tu empresa para que el lenguaje sea realmente unificado o estandarizado dentro de tu empresa.

Por Leticia Ortiz

¿Quieres aprender más sobre ingeniería de software y dirección de proyectos?

Regístrate GRATIS a nuestro boletín y recibe videos exclusivos, artículos, descuentos sustanciosos en cursos y mucho, mucho más...

Nombre

Email

¡REGÍSTRAME YA!

VER MÁS ARTÍCULOS

Contáctanos

Teléfono en la Ciudad de México:

+52 (55) 5594 6411
+52 (55) 5594 6415

E-mail: **cursos@abiztar.com.mx**

O si lo prefieres **llena el formulario de contacto**

Ver aviso de privacidad

Sigue a Abiztar en redes sociales

Entérate de promociones, descuentos, artículos, videos y más.

SÍGUENOS EN FACEBOOK SÍGUENOS EN TWITTER