Elementos de Arquitectura y Diseño

Teoría

Dominio del sistema

Qué es el dominio del sistema?

Cuando hablamos del dominio de un sistema, nos referimos a todo aquello que pertenece a la esfera del conocimiento relacionado con nuestro sistema. Esto incluye (y no se restringe a) los procesos involucrados en el sistema, los actores presentes en los mismos, y sus interacciones. Los roles en los que se clasifican a esos actores, y los eventos que suceden durante dichos procesos, también forman parte del dominio del sistema.

Ejemplo de dominio

En el dominio de un kiosco podemos encontrarnos con:

- Productos,
- Ventas,
- Stock,
- Falta de Stock,
- Clientes,
- Dinero,
- Reclamo,
- Devolución,
- etc

Modelos

Qué es un modelo?

Cuando hablamos de nuestra tarea como desarrolladores de software, lo que estamos haciendo es construir un modelo computacional de una porción de la realidad. Para ello es primordial comprender el dominio del sistema que se quiere modelar. Cada uno de los actores y componentes de ese sistema (mundo real) estará representado en mi sistema (computacional) con un modelo (representación de la realidad) de dicho componente. En los sistemas orientados a objetos, podemos pensar en un modelo como aquello que caracteriza o define a una familia de instancias presentes en mi sistema. En lenguajes con presencia de clases, podemos asimilar ambos conceptos bajo el mismo techo. Sin embargo, aquellos lenguajes que carecen de este tipo de clasificación (lenguajes sin clases) pueden presentar también tales representaciones entre sus componentes.

Ejemplo de modelo

Continuando con el ejemplo del kiosco, tomemos una venta, por ejemplo. Qué cosas caracterizan a una venta? Qué cosas esperamos encontrarnos cuando tratamos con una venta?

- En qué fecha se realizó
- Qué producto se vendió
- Qué cantidad de dicho producto

Sin embargo, estos detalles no son universales, y dependen ampliamente del sistema que estoy modelando! Ya que por ejemplo, quizás nos interesaría también conocer:

- Quién recibió el producto
- Si se entregó en mano o se envió por mensajería
- La forma de pago: efectivo? débito? crédito?
- etc

Una vez que un componente de nuestro sistema está definido (mediante el análisis funcional correspondiente) es preciso volcar en algún lugar de nuestro modelo toda esta información obtenida. Además, quisiéramos poder diferenciar instancias válidas de un modelo, de aquellas que no lo sean. Por ejemplo, podemos decir que sólo serán ventas válidas en nuestro sistema, aquellas en que:

- la fecha esté expresada con dia, número de mes, y los dos últimos digitos del año
- La cantidad de productos vendidos sea siempre un número entero positivo
- Etc

Es por esto que es altamente recomendable controlar minuciosamente la creación de instancias de dichos modelos antes de permitir que éstas circulen libremente por el sistema, sin control alguno.

Es una buena práctica, por lo tanto, encapsular la creación de las instancias de cada modelo junto con la validación de los datos que se utilizarán para la construcción de dicha instancia. Esto puede realizarse mediante la utilización de clases o de funciones constructoras. A estas instancias se las conoce como **Entidades**.

Entidades

Qué son las Entidades?

Una entidad dentro de un sistema representa una instancia de algún modelo. En particular, lo que distingue a las entidades de cualquier otro objeto es que éstas poseen un identificador único que las distingue entre sí. Es decir, a la hora de comparar dos entidades, no se compararán sus atributos sino su identificador.

Ejemplo de entidad

Supongamos que queremos representar usuarios en un sistema. Es posible que cada usuario tenga un nombre y apellido, una edad, una dirección, etc. Sin embargo, a la hora de distinguir entre dos usuarios, nunca compararemos sus nombres, o sus apellidos, ni sus edades (estos podrían coincidir, salvo que existiera alguna regla de negocio que nos obligase a que no haya repetidos), ni tampoco nos tomaremos el trabajo de comparar individualmente cada uno de sus campos para asegurarme de que sean instancias distintas. Simplemente alcanza con comparar sus identificadores únicos para decidir si se trata de la misma entidad o no. Esta característica diferencia a las entidades de los demás objetos no identificables que pueden existir circulando por el sistema. A estos últimos se los conoce como Objetos de Valor - Value Objects (VO).

Value Objects (VO)

Qué son los Value Objects?

A diferencia de las entidades, éstos son instancias de objetos cuya única manera de ser identificados es a través del valor de sus atributos. Esto permite que dos instancias creadas en distintos momentos y en diferentes contextos pero con los mismos valores asignados a sus atributos sean consideradas idénticas al ser comparadas, y sea imposible diferenciar una de la otra.

Ejemplo de Value Object

Algunos de los incontables ejemplos pueden ser:

- Cantidades de dinero: $1000 creados por mí son iguales a $1000 creados por vos.
- Colores: suponiendo un color en formato RGB, mi (255,10,8) es igual a tu (255,10,8)
- Fechas: dos personas que cumplen el mismo día, cada una tiene un objeto fecha asignado a sus atributos, sin embargo al compararlas, ambas fechas son idénticas.
- etc

Business Objects (BO)

Ok, ya modelé Entidades y Valores... Qué hay de los procesos del sistema?

Ya hemos hablado de que en un sistema se llevan a cabo procesos o tareas. Es decir, las actividades realizadas por/entre las entidades presentes, las cuales usualmente consisten en el intercambio y/o procesamiento de ciertos valores que fluyen libremente por el sistema. Es importante entonces poder modelar estos procesos. Existen varias maneras de atacar esta necesidad:

Una de ellas es pensar en que cada una de esas actividades tiene un responsable, es decir, un **actor** que las lleva a cabo. En ocasiones ese actor es una de las **entidades** presentes en el dominio, mientras que otras veces, ese actor es una entidad que existe únicamente en nuestro modelo computacional, con el único fin de asignarle a alguien la responsabilidad de realizar dicha actividad/tarea. La forma de asignar esas actividades a sus actores es en forma de **métodos** de la entidad. Se dice que estas entidades cargadas de métodos con lógica de negocio pasan a ser **Objetos de Negocio** o Business Objects (BO).

La segunda opción entre las más comunes es modelar directamente el proceso o **Caso de Uso** como un objeto en sí mismo. Es decir, un objeto que representa una **ocurrencia o realización de una tarea** determinada. **Una vez finalizada, el objeto se desecha**. Estos objetos carecen de un identificador o de atributos que los identifiquen, ya que su razón de ser no implica comparaciones entre distintos casos de uso, sino más bien su instanciación y ejecución. Al estar estar íntegramente compuestos por las reglas o lógica del negocio, estos objetos también son considerados **Objetos de Negocio**.

Data Transfer Objects (DTO)

Y qué hay de todos los demás objetos que circulan por el sistema?

Como ya hemos notado, la utilización de objetos como simples contenedores de datos es ampliamente utilizada para transferir información de un lado a otro del sistema. A veces precisamos la presencia de un cliente en nuestro sistema para que participe de una tarea que le corresponde realizar. De dónde vamos a sacar a ese cliente? Pues claro, de una base de datos (o de algún otro medio de persistencia de nuestra preferencia). Sin embargo, las bases de datos no pueden almacenar funcionalidad, solo datos. Existe entonces, en algún momento en el tiempo inmediatamente luego de recuperar al cliente desde la base de datos, un objeto que parecería ser una entidad, por sus atributos y su identificador, pero que sin embargo, carece de funcionalidad (por lo que más adelante se deberá instanciar la entidad correspondiente utilizando dichos datos recuperados). A estos objetos transportadores de datos se los conoce como Objetos Transportadores de Datos o **Data Transfer Objects** (DTOs). En algunos lenguajes (de tipado estático) es preciso crear clases para estos objetos, mientras que en otros (de tipado dinámico) no lo es, y sus características quedan plasmadas en el sistema en forma implícita.

Apéndice A: Errores

Qué son los errores?

Son eventos que ocurren durante un proceso, que interrumpen el orden natural de las tareas en desarrollo.

Los errores, por lo tanto no son meros elementos del código, ni herramientas usadas por el equipo de desarrollo, sino que son eventos que **forman parte del dominio del sistema** que estamos modelando.

Dentro de los sistemas de administración podemos encontrar, generalmente, algunos errores recurrentes. Los más frecuentes son aquellos relacionados con las operaciones de alta, baja, modificación y consulta (o en ingles CRUD: create, read, update, delete).

Ejemplo de error

Ante un pedido de **creación de un cliente**, podemos encontrarnos con el requisito de que ésta sea mayor de edad. Por lo tanto, requerir la creación de tal recurso proporcionando una edad menor a la requerida estaría en directa violación de una política del negocio. En este caso, podríamos encontrarnos con el siguiente error:

- Error por edad menor a la requerida

En un servidor REST esta operación se realiza normalmente mediante una petición de tipo POST, y el la respuesta que genera se identifica con un código de estado 400 (BAD REQUEST), sin embargo el código de estado pertenece al protocolo HTTP, lo cual es un detalle de implementación de la capa de ruteo, y no forma parte del dominio del problema. Es decir, el número 400 no representa nada dentro de los procesos de la empresa, y por lo tanto no debe estar presente tampoco en los modelos del dominio.