Apuntes iniciales. TP1 TADP

Paradigmas Orientado a Aspectos

El principal objetivo de la POA es la separación de las funcionalidades dentro del sistema:

- Por un lado funcionalidades comunes utilizadas a lo largo de la aplicación.
- Por otro lado, las funcionalidades impropias de cada módulo.

Cada funcionalidad común se encapsulará en una entidad.

- **Aspecto:** es una funcionalidad transversal (*cross-cutting*) que se va a implementar de forma modular y separada del resto del sistema. El ejemplo más común y simple de un aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso.
- **Punto de Cruce o de Unión:** es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.
- **Consejo:** es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.
- **Puntos de Corte:** define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.
- **Introducción:** permite añadir métodos o atributos a clases ya existentes. Un ejemplo en el que resultaría útil es la creación de un Consejo de Auditoría que mantenga la fecha de la última modificación de un objeto, mediante una variable y un método setUltimaModificacion(fecha), que podrían ser introducidos en todas las clases (o sólo en algunas) para proporcionarles esta nueva funcionalidad.
- **Destinatario:** es la clase aconsejada, la clase que es objeto de un consejo. Sin AOP, esta clase debería contener su lógica, además de la lógica del aspecto.
- **Resultante:** es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP).
- **Tejido:** es el proceso de aplicar Aspectos a los Objetos Destinatarios para crear los nuevos Objetos Resultantes en los especificados Puntos de Cruce. Este proceso puede ocurrir a lo largo del ciclo de vida del Objeto Destinatario:
  - Aspectos en Tiempo de Compilación, que necesita un compilador especial.
  - Aspectos en Tiempo de Carga, los Aspectos se implementan cuando el Objeto Destinatario es cargado. Requiere un ClassLoader especial.
  - Aspectos en Tiempo de Ejecución.

La programación orientada a aspectos (POA) es una nueva metodología de programación que aspira a soportar la separación de competencias para los aspectos antes mencionados. Es decir, que intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema. En definitiva, lo que se persigue es implementar una aplicación de forma eficiente y fácil de entender.

La dificultad de modularizar estas incumbencias, se debe a que las técnicas tradicionales de programación proveen la posibilidad de descomponer el problema de acuerdo a una única dimensión (la dominante), permitiendo un efectiva modularización de las incumbencias de dicha dimensión a costa de sacrificar una buena modularización de las incumbencias de las demás dimensiones. Este problema ha sido denominado “la tiranía de la descomposición dominante” y como consecuencia del mismo las incumbencias de las dimensiones no dominantes adquieren una naturaleza transversal que les otorga el nombre de incumbencias transversales.

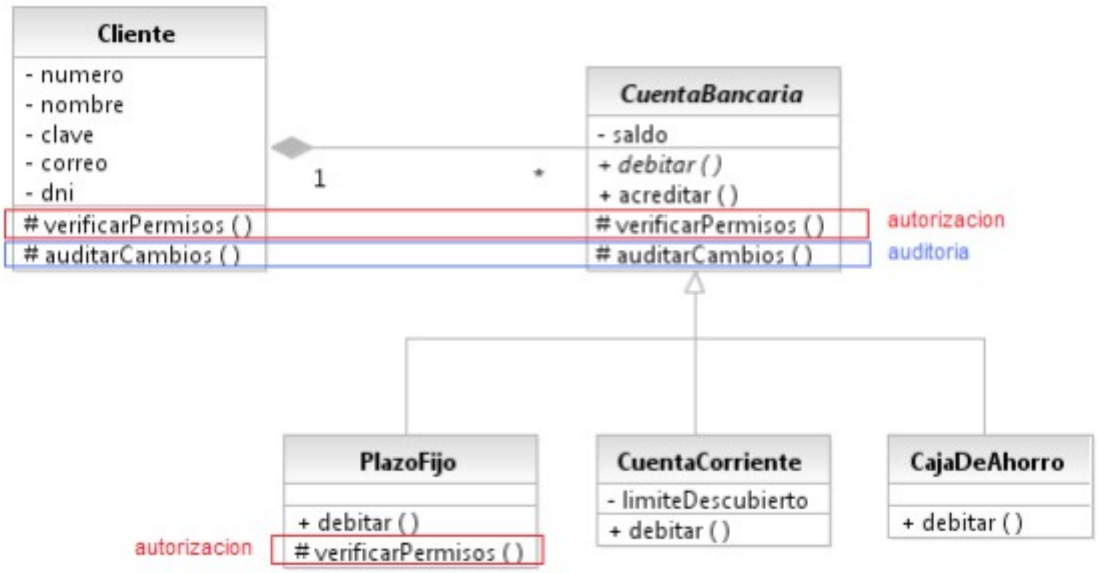

Figura 2: Incumbencias transversales en las clases del dominio de banca electrónica

Conceptos y explicación interesantes:
El paradigma asimétrico por su parte, hace una diferenciación entre las incumbencias dominantes (también llamadas centrales o funcionales) las cuales modela mediante componentes base que luego son compuestos con las incumbencias no dominantes (también llamadas aspectos, no centrales o secundarias). Como se vio en el ejemplo de la sección anterior, las incumbencias no centrales en un paradigma asimétrico, tienen generalmente una naturaleza transversal. El exponente más representativo de este paradigma es la programación orientada a aspectos (aspect oriented programming, AOP).

La mezcla de los componentes base y los aspectos da como resultado la aplicación final. Dicha mezcla se realiza por medio de un proceso denominado entretejido (weaving). La asimetría entre los componentes base y los aspectos pone de manifiesto en enfoque asimétrico de AOP en lo que a separación de incumbencias respecta.

Es importante destacar las dos propiedades de AOP mencionadas en el párrafo anterior: la transparencia y la cuantificación. La transparencia tiene que ver con la posibilidad de introducir aspectos en el código base de forma transparente, de manera tal que quien vea el código base no pueda predecir la existencia de los aspectos. Esta propiedad puede resultar polémica, pero es justamente una de las propiedades que distinguen a AOP de las técnicas predecesoras. Por su parte, la cuantificación brinda la posibilidad de expresar enunciados (statements) del tipo: En todo programa P, cuando se dé la condición C, ejecutar la acción A. El programa P no es más que un programa tradicional constituido por componentes base, mientras que la acción A es parte de un aspecto que será entretejido con el programa P en el punto indicado por la condición C. La

cuantificación puede ser estática, si la condición C trata sobre la estructura del programa P, o bien dinámica, si la condición C tiene que ver con algún suceso de la ejecución de P. Al mismo tiempo, la cuantificación estática puede ser de caja negra (cuantificación sobre los elementos de la interfase pública del programa P) o de caja blanca (cuantificación sobre la estructura interna del programa P).

Joinpoints:
Algunos ejemplos de joinpoints son: una llamada a un método, la creación de una instancia, el manejo de una excepción, la ejecución de un ciclo, el retorno de un método, la asignación de un valor a una variable, la modificación de un atributo, entre otros.

Pointcut:
A un conjunto definido de joinpoints se lo llama pointcut. A diferencia de los jointpoints, los pointcuts, son definidos por el programador.
Todas las herramientas AOP brindan un conjunto de designadores de pointcuts que en conjunto con el uso de expresiones regulares permiten la definición de los pointcuts.

Advice:
El código del aspecto que se ejecuta asociado a un determinado pointcut es denominado advice. Conceptualmente los advices contienen el comportamiento del aspecto. Los advices puede ser básicamente de 3 tipos.
◆ Before: se ejecutan previos a la ejecución del joinpoint asociado.
◆ After: se ejecutan después de la ejecución del joinpoint asociado.
◆ Around: se ejecutan antes y después del joinpoint asociado, pudiendo incluso reemplazar la ejecución del joinpoint.

Declaraciones de intertipo:
Otro elemento de AOP, muchas veces pasado por alto, es la declaración de intertipos (también llamada introducción o mixin) que permite extender la estructura de un componente base mediante el agregado de miembros (métodos y atributos) e interfases.

Aspecto:
Un aspecto es una entidad que modulariza una incumbencia transversal mediante la definición de un conjunto de pointcuts, advices y declaraciones de intertipo.

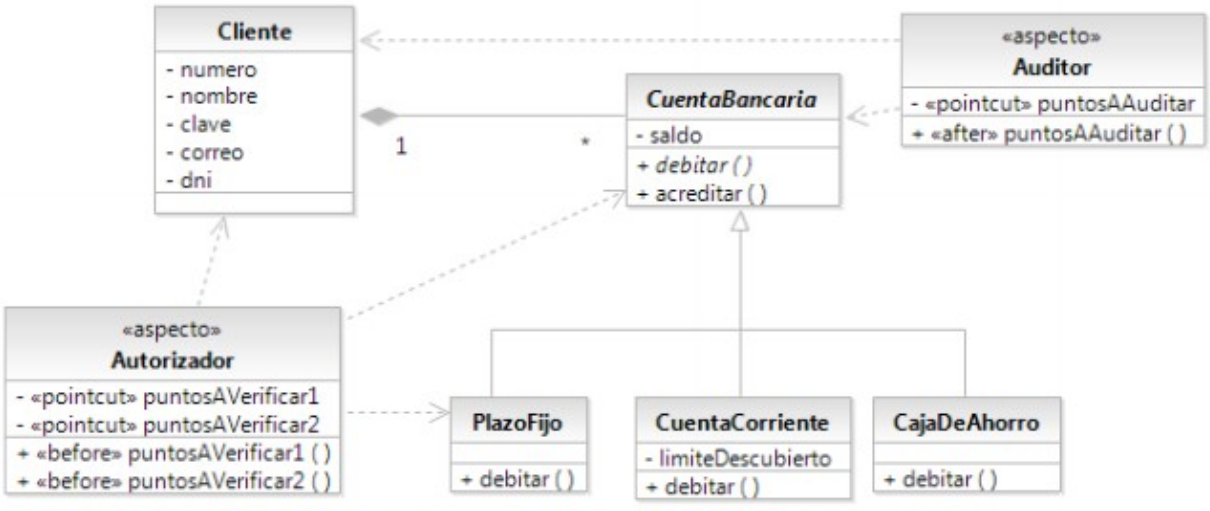

Figura 4: Diagrama de la solución AOP del dominio de banca electrónica

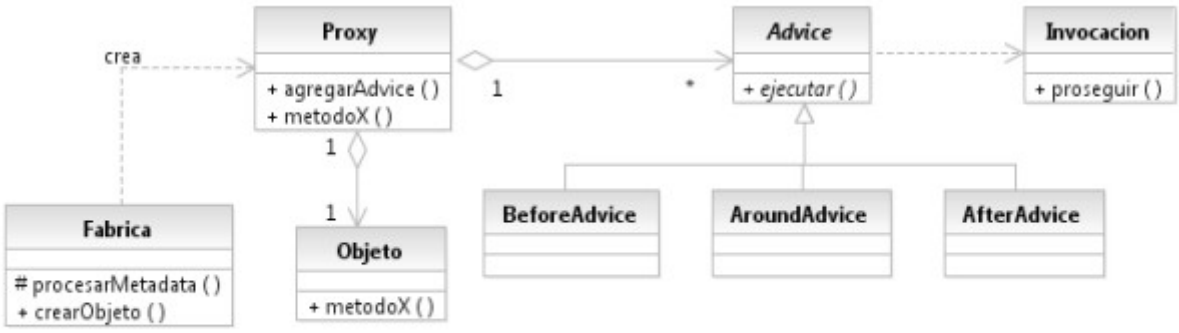

Figura 7: Clases típicas de un frameworks AOP.

La figura 7 muestra las clases típicas de un framework AOP, hay que destacar que la clase Proxy suele ser creada en tiempo de ejecución. Esta fábrica de objetos interpreta los pointcuts definidos en la metadata y en base a eso a la hora de crear un objeto detecta si deben agregarse aspectos al objeto creado, en cuyo caso crea un proxy con el conjunto de advices que aplicarán al objeto en cuestión y devuelve una referencia al proxy en lugar de una referencia directa el objeto pedido. El diagrama de secuencia de la figura 8 ilustra este proceso

Expresiones Regulares

Una **expresión regular**, a menudo llamada también ***regex***, es una secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Alternación

Una barra vertical separa las alternativas. Por ejemplo, "marrón|castaño" se corresponde con *marrón* o *castaño*.

Cuantificación

Un cuantificador tras un carácter especifica la frecuencia con la que éste puede ocurrir. Los cuantificadores más comunes son ?, + y *:

?

El signo de interrogación indica que el carácter que le precede puede aparecer como mucho una vez. Por ejemplo, "ob?scuro" se corresponde con *oscuro* y *obscuro*.

+

El signo más indica que el carácter que le precede debe aparecer al menos una vez. Por ejemplo, "ho+la" describe el conjunto infinito *hola*, *hoola*, *hooola*, *hoooola*, etcétera.

*****

El asterisco indica que el carácter que le precede puede aparecer cero, una, o más veces. Por ejemplo, "0*42" se corresponde con *42*, *042*, *0042*, *00042*, etcétera.

Agrupación

Los paréntesis pueden usarse para definir el ámbito y precedencia de los demás operadores. Por ejemplo, "(p|m)adre" es lo mismo que "padre|madre", y "(des)?amor" se corresponde con *amor* y con *desamor*.

El punto ".

El punto se interpreta por el motor de búsqueda como "cualquier carácter", es decir, busca cualquier carácter SIN incluir los saltos de línea.

Ejemplo: Si se le dice al motor de RegEx que busque "g.t" en la cadena "el gato de piedra en la gótica puerta de getisboro goot" el motor de búsqueda encontrará "gat", "gót" y por último "get".

La barra inversa o antibarra "\"

Se utiliza para "marcar" el siguiente carácter de la expresión de búsqueda de forma que este adquiera un significado especial o deje de tenerlo. O sea, la barra inversa no se utiliza nunca por sí sola, sino en combinación con otros caracteres. Al utilizarlo por ejemplo en combinación con el punto "\." este deja de tener su significado normal y se comporta como un carácter literal.

Los corchetes "[ ]"

La función de los corchetes en el lenguaje de las expresiones regulares es representar "clases de caracteres", o sea, agrupar caracteres en grupos o clases. Son útiles cuando es necesario buscar uno de un grupo de caracteres. Dentro de los corchetes es posible utilizar el guion "-" para especificar rangos de caracteres. Adicionalmente, los metacaracteres pierden su significado y se convierten en literales cuando se encuentran dentro de los corchetes.

Ejemplo: La expresión regular "[\dA-Fa-f]" nos permite encontrar dígitos hexadecimales.

El signo de dólar "$"

Representa el final de la cadena de caracteres o el final de la línea, si se utiliza el modo multi-línea. No representa un carácter en especial sino una posición. Si se utiliza la expresión regular "\.$" el motor encontrará todos los lugares donde un punto finalice la línea, lo que es útil para avanzar entre párrafos.

El acento circunflejo "^"

Este carácter tiene una doble funcionalidad, que difiere cuando se utiliza individualmente y cuando se utiliza en conjunto con otros caracteres especiales. En primer lugar su funcionalidad como carácter individual: el carácter "^" representa el inicio de la cadena (de la misma forma que el signo de dólar "$" representa el final de la cadena). Por tanto, si se utiliza la expresión regular "^[a-z]" el motor encontrará todos los párrafos que den inicio con una letra minúscula.

Las llaves "{}"

Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular de la siguiente forma: "\d{2}". Esta expresión le dice al motor de búsqueda que encuentre dos dígitos contiguos.