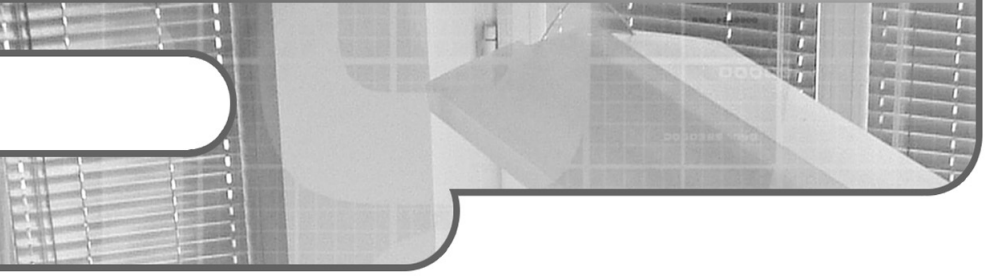

Capítulo 12
El patrón de diseño Composite

1. Descripción

El objetivo del patrón de diseño `Composite` es materializar composiciones de objetos en forma de estructura de árbol. Estas composiciones están concebidas de tal forma que un cliente tratará indiferentemente un componente y un compuesto.

2. Ejemplo

En nuestro sistema de venta de vehículos, queremos representar las empresas cliente, en especial para conocer el número de vehículos de los que disponen y proporcionarles ofertas de mantenimiento para su parque de vehículos.

Las empresas que posean filiales solicitan ofertas de mantenimiento que tengan en cuenta el parque de vehículos de sus filiales.

Una solución inmediata consiste en procesar de forma diferente las empresas sin filiales y las que posean filiales. No obstante esta diferencia en el procesado entre ambos tipos de empresa haría que la aplicación fuera más compleja y totalmente dependiente de la composición interna de las empresas cliente.

El patrón de diseño `Composite` resuelve este problema unificando ambos tipos de empresa y utilizando la composición recursiva. Esta composición recursiva es necesaria puesto que una empresa puede tener filiales que posean, ellas mismas, otras filiales. Se trata de una composición en árbol tal y como se ilustra en la figura 12.1 donde las empresas madre se sitúan sobre sus filiales. Por razones evidentes de simplificación, suponemos que no hay filiales comunes a dos empresas.

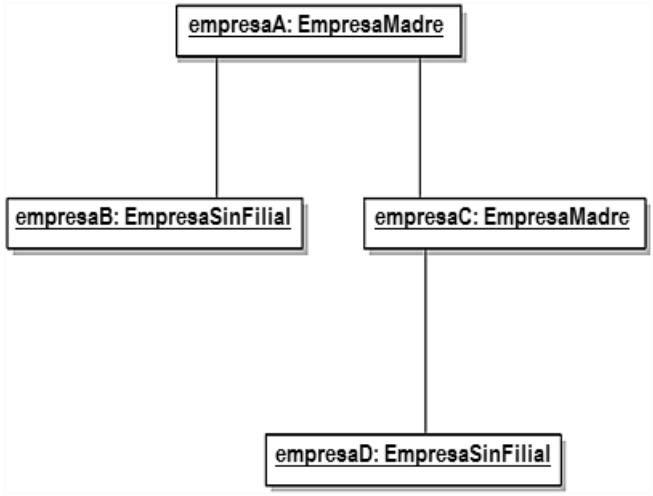

Figura 12.1 - Árbol de empresas madres y de sus filiales

La figura 12.2 muestra el diagrama de clases correspondiente. La clase abstracta `Empresa` contiene la interfaz destinada a los clientes. Posee dos subclases concretas, a saber `EmpresaSinFilial` y `EmpresaMadre`, esta última guarda una relación de agregación con la clase `Empresa` representando los enlaces con sus filiales.

© Editions ENI - All rights reserved

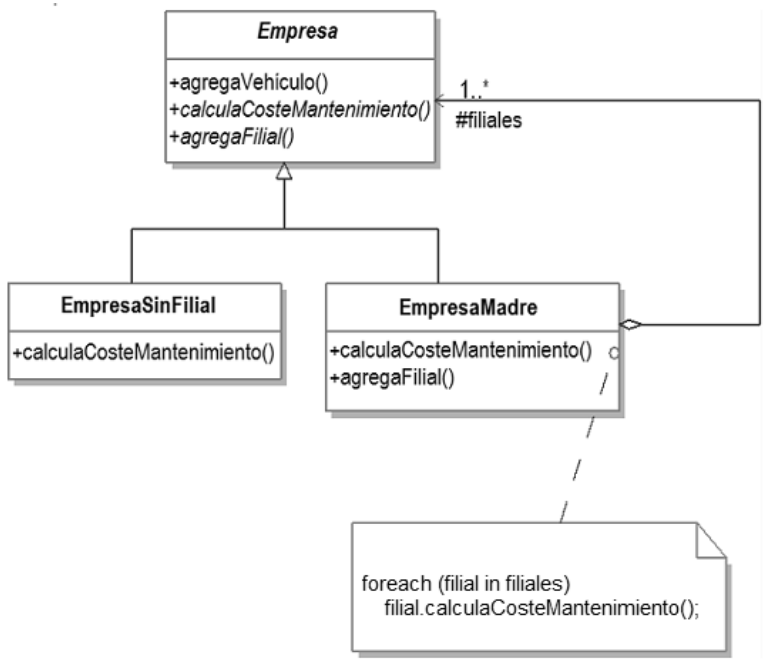

Figura 12.2 - El patrón de diseño `Composite` aplicado a la representación de empresas y sus filiales

La clase `Empresa` posee tres métodos públicos de los cuales dos son abstractos. El método concreto es el método `agregaVehiculo` que no depende de la composición en filiales de la empresa. En cuanto a los otros dos métodos, se implementan en las subclases concretas (`agregaFilial` no tiene implementación en `EmpresaSinFilial`, por eso no se representa en el diagrama de clases).

3. Estructura

3.1 Diagrama de clases

La figura 12.3 detalla la estructura genérica del patrón de diseño `Composite`.

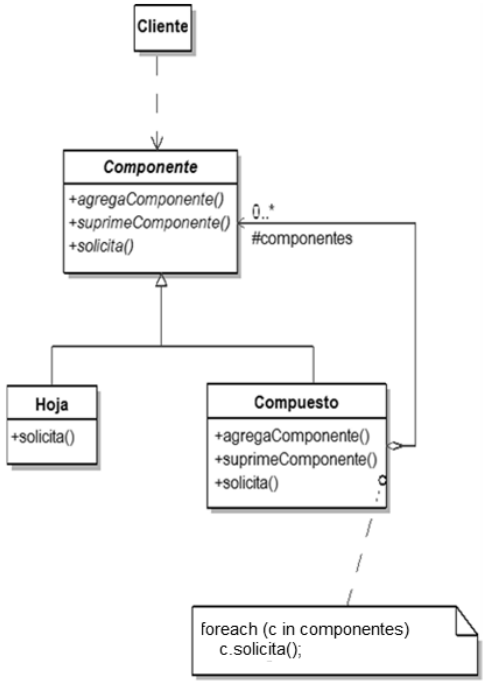

Figura 12.3 - Estructura del patrón de diseño `Composite`

3.2 Participantes

Los participantes del patrón de diseño `Composite` son los siguientes:

– `Componente` (`AbstractEmpresa`) es la clase abstracta que describe la interfaz de los objetos de la composición, implementa los métodos comunes e introduce la firma de los métodos que gestionan la composición agregando o suprimiendo componentes.

© Editions ENI - All rights reserved

– `Hoja` (`EmpresasSinFilial`) es la clase concreta que representa las hojas de la composición (en una estructura de árbol, una hoja no posee componentes, es un nudo terminal).
– `Compuesto` (`EmpresaMadre`) es la clase concreta que representa los objetos compuestos de la jerarquía. Esta clase posee una asociación de agregación con la clase `Componente`.
– `Cliente` es la clase de los objetos que acceden a los objetos de la composición y los manipulan.

3.3 Colaboraciones

Los clientes envían sus peticiones a los componentes a través de la interfaz de la clase `Componente`.

Cuando un componente recibe una petición, reacciona en función de su clase. Si el componente es una hoja, procesa la petición tal y como se ilustra en la figura 12.4.

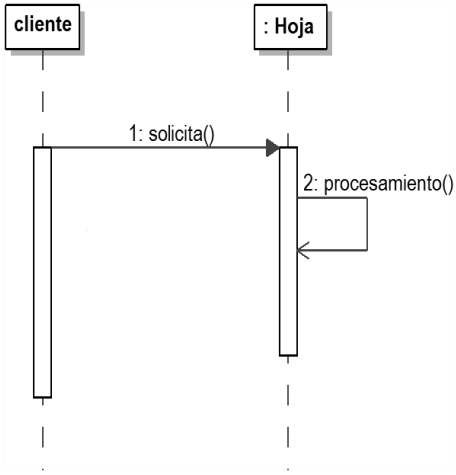

Figura 12.4 - Procesado de un mensaje por parte de una hoja

Si el componente es una instancia de la clase `Compuesto`, realiza un procesado previo, luego generalmente envía un mensaje a cada uno de sus componentes y realiza un procesamiento posterior. La figura 12.5 ilustra este comportamiento de llamada recursiva a otros componentes que van a procesar, a su vez, esta petición bien como hoja o bien como compuesto.

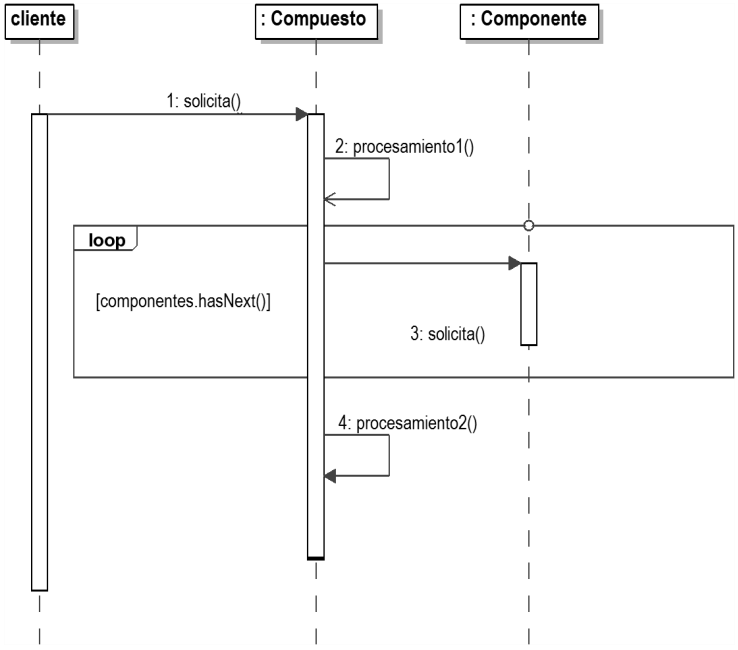

Figura 12.5 - Procesado de un mensaje por parte de un compuesto

4. Dominios de aplicación

El patrón de diseño Composite se utiliza en los siguientes casos:

– Es necesario representar jerarquías de composición en un sistema.

– Los clientes de una composición deben ignorar si se comunican con objetos compuestos o no.

© Editions ENI - All rights reserved

5. Ejemplo en PHP

Retomemos el ejemplo de las empresas y la gestión de su parque de vehículos.

El código fuente en PHP de la clase abstracta `AbstractEmpresa` se describe a continuación. Conviene observar que el método `agregaFilial` devuelve un resultado booleano que indica si ha sido posible realizar o no la agregación.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Composite;

abstract class AbstractEmpresa
{
   protected static float $costeUnitarioVehiculo = 5;

   protected int $numVehiculos = 0;

   public function agregaVehiculo(): void
   {
       $this->numVehiculos++;
   }

   abstract public function calculaCosteMantenimiento(): float;

   abstract public function agregaFilial(AbstractEmpresa
$filial): bool;
}
?>
```

El código fuente de la clase `EmpresaSinFilial` aparece a continuación. Lógicamente, las instancias de esta clase no pueden agregar filiales.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Composite;

class EmpresaSinFilial extends AbstractEmpresa
{
   public function agregaFilial(AbstractEmpresa $filial): bool
```