

Meta-Patrones:

Una nueva aproximación a los patrones de diseño.

Moisés Daniel Díaz Toledano.

Email : moises@moisesdaniel.com

WebSite : <http://www.moisesdaniel.com>

1.0.- Patrones de diseño.

La noción de 'patrón de diseño' existía con mucha anterioridad a su uso común en la comunidad informática. Como muestra de ello, existen numerosos trabajos que así lo demuestran [6]. Sin embargo es comúnmente aceptado que el trabajo que dió origen a esta disciplina tal y como hoy la entendemos es el GOF [8][3].

Existen numerosas definiciones de 'patrón'. Una de ellas es la acuñada por Richard Gabriel [1][7] y que dice lo siguiente: "*Cada patrón es una regla con 3 partes, la cual expresa la relación entre un contexto concreto, un sistema de fuerzas que ocurren repetidamente en dicho contexto, y una configuración software específica que permite que estas fuerzas se resuelvas por ellas mismas*". También podemos aceptar la siguiente: "*Un patrón es la representación abstracta de una buena solución a un problema concreto y generalmente frecuente que ocurre en uno o más contextos*".

Por lo general, los patrones se estructuran en 'Lenguajes de Patrones', pudiéndose definir éstos como: *"La especificación de una serie de roles (patrones) y sus relaciones (patrones) que nos permiten describir buenas soluciones a los diferentes problemas que aparecen en un contexto específico"*.

El objetivo de los patrones de *diseño* es el de capturar buenas prácticas que nos permitan mejorar la calidad del diseño de un sistema, *determinando objetos* que soporten roles útiles en dicho contexto, *encapsulando complejidad*, y *haciéndolo más flexible*

Podemos observar que la estructura de estas soluciones (patrones) se repite, utilizando una serie de mecanismos básicos (incluso en diferentes niveles de abstracción) para producir los mismos efectos en el sistema. Conocer estos mecanismos comunes nos permite tener una visión más clara y con cierta perspectiva sobre los patrones que se nos van presentando, así como tener la capacidad de *generarlos*.

2.0.- Meta-Patrones.

Todos queremos construir software correcto, robusto, extensible y reusable. Debemos ser conscientes que este conjunto de factores externos de calidad guarda una relación directa con la calidad interna del mismo, es decir con su corrección, flexibilidad, etc. Sin embargo...

¿Cómo conseguimos corrección? Uno de los mecanismos fundamentales consiste en reducir la complejidad inherente al problema, encapsulando abstracciones así como normalizando y uniformando.

¿Cómo conseguimos flexibilidad? Básicamente añadiendo indirección (desacoplando abstracciones). ¿Dónde añadimos esta indirección? En el lugar en el que se espere la evolución del sistema o ya es necesitada.

2.1.- Diagrama Objetivo-Mecanismo.

Podemos visualizar todo esto en un diagrama de texto al que llamaremos Diagrama Objetivo-Mecanismo y que representa varios niveles de abstracción:

Diagrama:

- Software correcto.

?? Divido lo complejo (hago las cosas simples).

- o Encapsulo, abstraigo: Funciones, clases, componentes, aplicaciones, sistemas (básicamente estructural). ¿Cómo conseguimos la simplicidad en?

~~///~~ Funciones: Unicidad funcional, mismo nivel semántico en su discurso.

- /// Clases: unicidad identificativa, evitar desnivel semántico (métodos que traten abstracciones similares, relaciones de nivel abstracto similar).
- ///
- o **Uniformo** – normalizo (básicamente funcional es funcional).
 - /// Código: Notación, mecanismos de validación (o blindaje), mecanismos de inicialización, mecanismos de tratamiento de errores, sangrado.
 - /// Niveles de abstracción uniforme en los discursos (programación), interfaces.
 - /// Mecanismos de Interacción (también visualización).
 - /// Mecanismos de comunicación entre entidades de similares niveles de abstracción . Uso los mismos ‘métodos’ para hacer las mismas cosas (componentes, etc).

- Software flexible-reutilizable.

?? Añado Indirección (desacoplo).

- o Creo un elemento intermedio (gano capacidad de manipulación, etc también me alejo más del problema en sí mismo, y constituye otro elemento a mantener). Estos elementos intermedios pueden ser:
 - /// Función.
 - /// Clase.
 - /// Clases + Polimorfismo.
 - /// Componente.
 - /// Subsistema.
 - /// Etc...

En un principio podemos entender estos Objetivos-Mecanismos como generadores de patrones ya que como voy a especificar más abajo nos ayudan a crear patrones de diseño. Por esta razón los denominaré Meta-Patrones, pasando a especificarlos dentro del contexto de los patrones de diseño orientado a objetos.

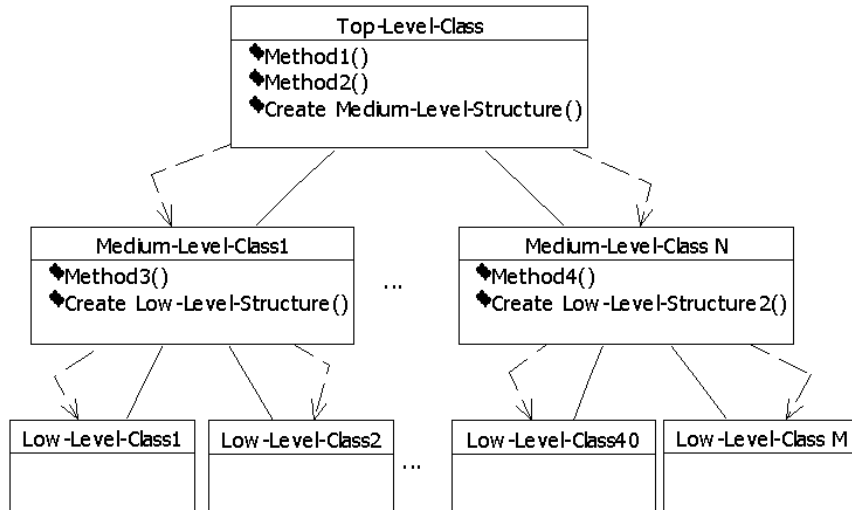
2.2.- Meta-Patrón Encapsulador (encapsulador).

- Problema: Tenemos cierta funcionalidad o interacción compleja distribuida entre varios objetos.

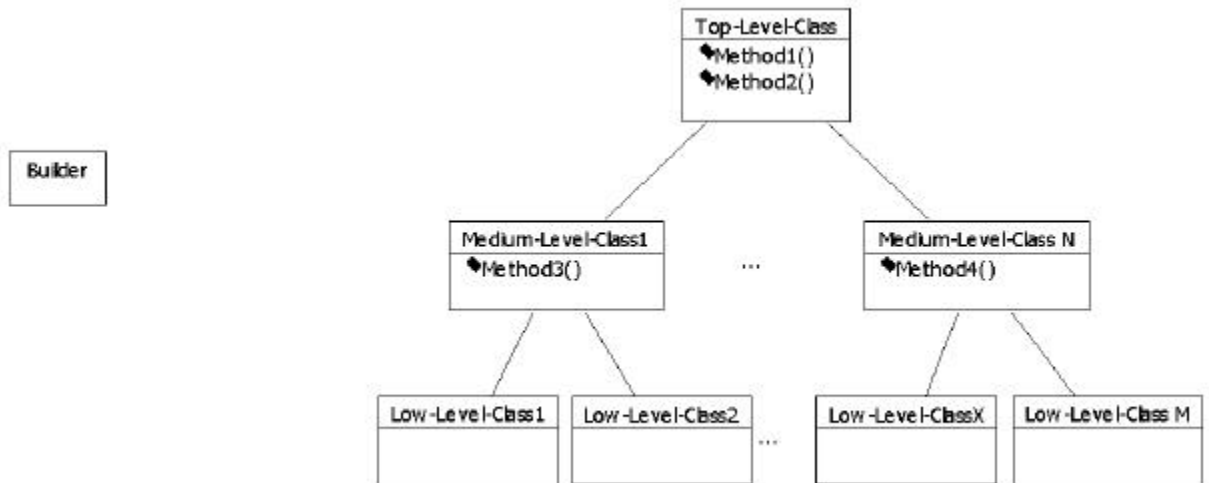
- Solución: Encapsular dicha funcionalidad o interacción en un objeto (dependiendo del nivel de abstracción en el que nos encontremos podría estar en una función, clase, componente, subsistema, etc.) El encapsulador provee simplificación sobre el diseño generando un nuevo rol.

- Problema de ejemplo: La creación de una compleja estructura de objetos (Builder).

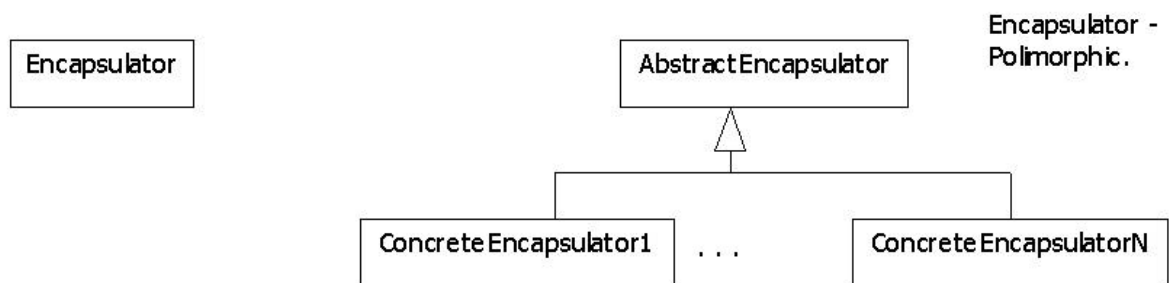
Estructura Original:



Solución Específica:



- Representación abstracta: Es una clase.

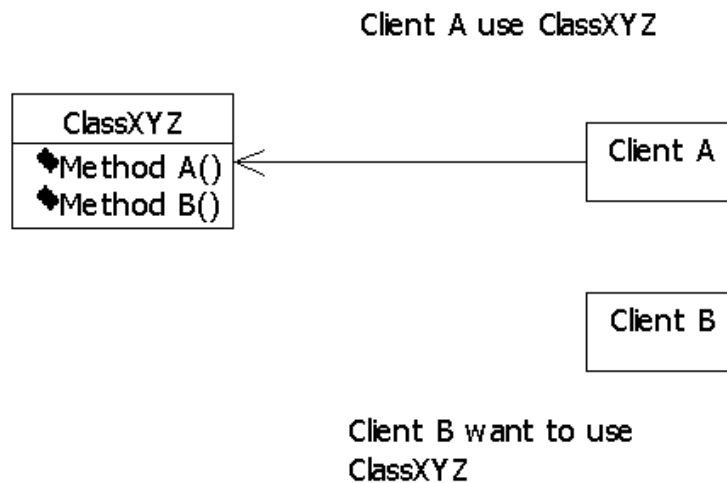


- Algunos patrones que lo implementan: Builder, Fachada, Iterador, Mediator, ...

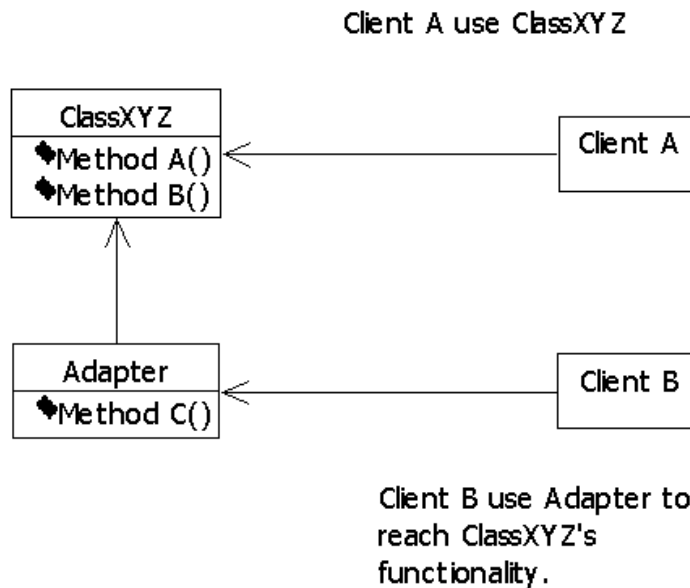
2.3.- Meta-Patrón Adder (Añadido).

- Problema: Tenemos una clase a la que queremos añadir cierta funcionalidad (en el sentido amplio de la palabra)
- Solución: Añadimos otra clase en la que encapsulamos dicha funcionalidad. El Adder provee nueva funcionalidad al sistema a través de un nuevo rol que hace de intermediador.
- Problema de ejemplo:

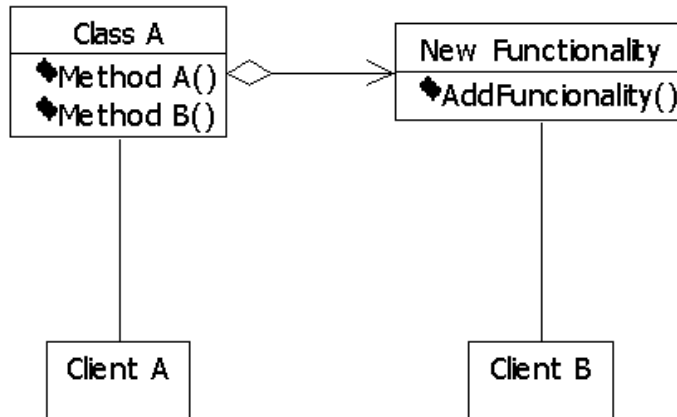
Estructura Original:



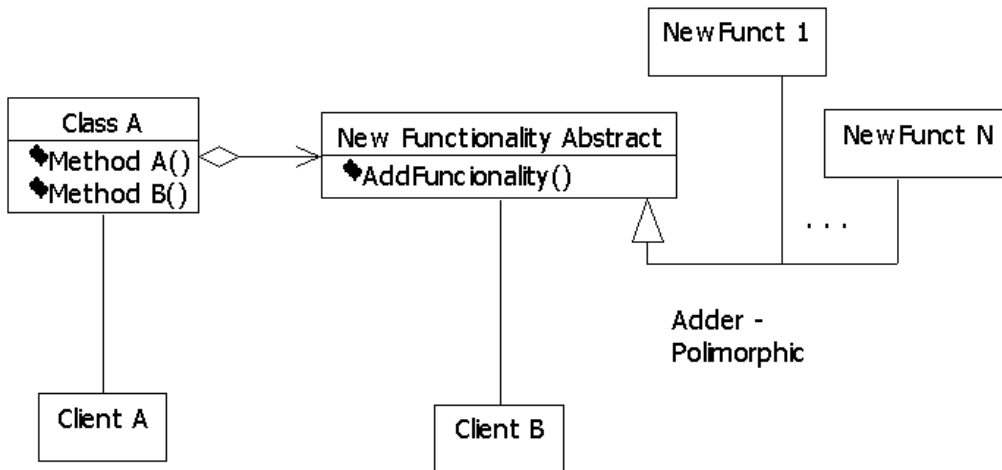
Solución específica:



- Representación abstracta:



Con Polimorfismo:



- Algunos patrones que lo implementan: Adaptador (añade una nueva interfaz), Proxy (añade control sobre el acceso a un objeto), comando (añadimos capacidad de control sobre la ejecución de métodos), decorator ...

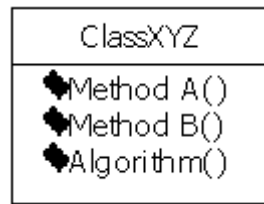
2.4.- Meta-Patrón Decoupler (desacoplador).

- Problema: Dentro de una clase existe cierta funcionalidad compleja que queremos flexibilizar o desacoplar para mejorar la calidad del diseño.

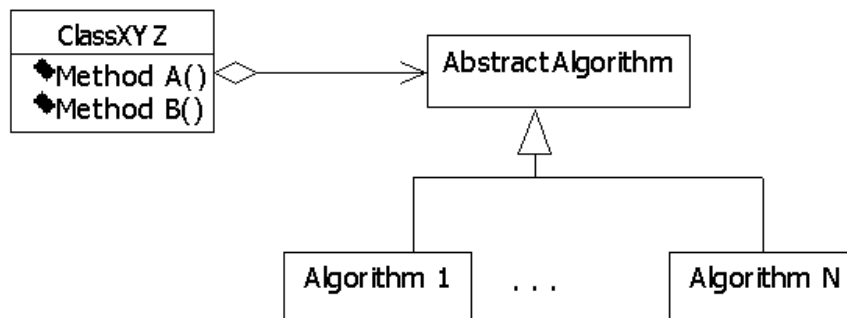
- Solución: Encapsulamos dicha funcionalidad en una clase agregada a la original, y la flexibilizamos aplicando polimorfismo. El Decoupler provee flexibilidad a alguna funcionalidad ya existente a través de un nuevo rol.

- Problema de ejemplo:

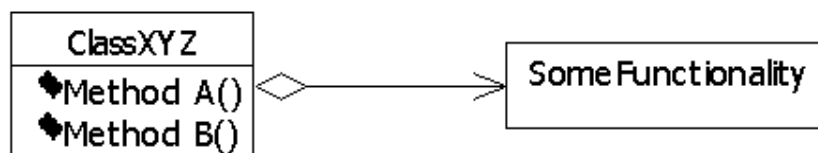
Estructura Original:



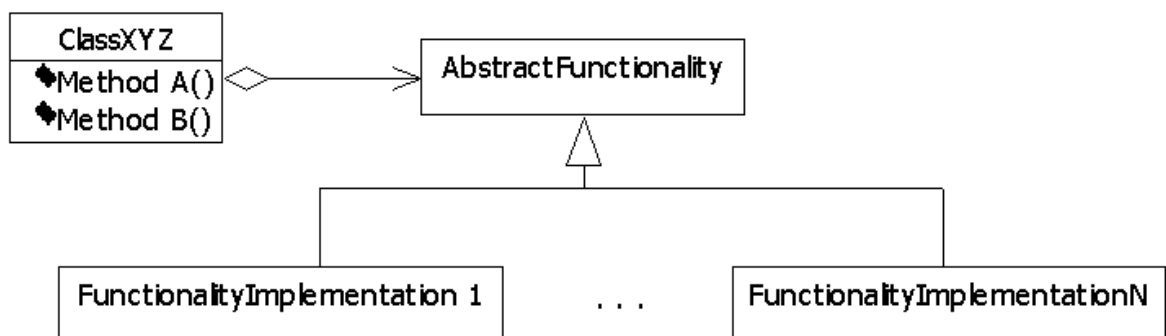
Solución específica:



- Representación abstracta:



Con polimorfismo



- Algunos patrones que lo implementan: Puente (desacopla la implementación), Estado (desacopla el estado de un objeto (conducta)), estrategia (desacopla algoritmo), ...

Estos Meta-Patrones o mecanismos, están presentes en la estructura de casi todos los patrones. Los únicos patrones no cubiertos (o que no pueden ser generados) por medio de los Meta-Patrones son los patrones que centran su utilidad en la adición de nuevos roles a un diseño (sin correspondencia previa con los elementos previamente existentes en dicho sistema), ya que aunque pueden verse como un 'Adder', su semántica es algo diferente ya que no encapsulan nada que ya estuviera presente en el sistema (podríamos verlos como el cuarto Meta-Patrón, y desde luego el más general). Estos objetos son bajo mi punto de vistas nuevos roles (**Common Roles**) . Encontrar, especificar y comunicar nuevos Common Roles dentro de contextos concretos es uno de los objetivos principales de los patrones de diseño que constituyen lenguajes de patrones. Estos nuevos roles no se generan por influencia del análisis del contexto de la aplicación, sino que generalmente suponen añadir cierta funcionalidad que se demuestra útil en dicho contexto.

En la mayoría de los patrones podemos ver que se encuentran uno o incluso varios meta-patrones.

En la siguiente sección se pueden ver algunos patrones de diseño generados de forma muy sencilla usando el concepto de metapatrón.

3.0.- Algunos patrones de ejemplo.

Con el concepto de metapatrón no es sólo más fácil entender los patrones de diseño sino el generarlos en nuestros diseños.

Aquí describo algunos patrones consecuencia directa del concepto de metapatrón.

3.1.- Validator.

- ?? Problema: Tenemos un objeto que tiene que hacer complejas validaciones con los argumentos que recibe y sobre los objetos con los que se relaciona.
- ?? Solución: Desacoplamos esta funcionalidad en otra clase (Validator) para hacer más sencillo, mantenible, etc al objeto original.
- ?? Estructura: Meta-Patrón Decoupler en el que encapsulamos las validaciones que debe de hacer el objeto. Si necesitamos más flexibilidad podríamos evolucionar hacia una estructura con polimorfismo (decoupler-polimorphic).

3.2.- Persistor (persistencia)

- ?? Problema: Tenemos en una clase implementada su persistencia en fichero, y necesitamos mayor flexibilidad (Base de Datos locales, remotas, etc).
- ?? Solución: Desacoplamos los mecanismos de persistencia de dicha clase.
- ?? Estructura: Meta-Patrón Decoupler - polimorphic en el que desacoplamos y encapsulamos los mecanismos de persistencia.

3.3.- Set.

- ?? Problema: Tenemos una gran colección de objetos (generalmente del mismo tipo) y tenemos que habilitar una manera sencilla de proveer información sobre dicho conjunto.
- ?? Solución: Damos origen a un nuevo Rol: Mapeador. Estos crean una estructura con todos los objetos que se deben mapear para tener un único punto de acceso a los mismos y centraliza la información del conjunto de estos objetos.
- ?? Estructura: Meta-Patrón Adder.

3.4.- Logger (patrón log).

- ?? Problema: Queremos tener un registro detallado de cómo funciona el sistema, con lo que obtenemos una estructura en la que el rol de mantener un registro del funcionamiento del sistema está distribuido.
- ?? Solución: Encapsulamos toda la funcionalidad de registrar información en una clase Log.
- ?? Estructura: Meta-Patrón Encapsulator.

Referencias:

- ?? [1] Pattern Definition Thread (<http://c2.com/cgi/wiki?PatternDefinionThread> ,
<http://c2.com/cgi/wiki?MorePatternDefinionThread>)
- ?? [2] A Pattern Definition, Software Patterns by James O. Coplien Bell
Laboratories Naperville, Illinois (<http://hillside.net/patterns/definition.html>)
- ?? [3] GOF (en la red) (<http://hillside.net/patterns/books//DPBook/DPBook.html>)
- ?? [4] A "Patterns BookList" on the WikiWiki Web:
<http://c2.com/cgi/wiki?BookList>
- ?? [5] The Patterns Home Page <http://hillside.net/patterns/patterns.html>
- ?? [6] Patter Origins and History http://www.enteract.com/~bradapp/docs/patterns-nutshell.html#Pattern_History
- ?? [7] A Pattern Language: Towns, Buildings, Construction (APL) Christopher
Alexander; Oxford University Press, 1977
- ?? [8] Design Patterns: Elements of Reusable Object-Oriented Software (GoF)
Gamma, Helm, Johnson, Vlissides; Addison-Wesley, 1994
- ?? [9] Pattern-Oriented Software Architecture: A System of Patterns (POSA)
Buschmann, Meunier, Rohnert, Sommerlad, Stal; Wiley and Sons, 1996
- ?? [10] Pattern Languages of Program Design (PLoPD1) Coplien and Schmidt
(editors); Addison-Wesley, 1995
- ?? [11] Plop conferences documents, <http://jerry.cs.uiuc.edu/~plop>
- ?? [12] Patterns Mailing Lists, <http://www.hillside.net/patterns/Lists.html>