PONTIFICIA UNIVERSIDAD JAVERIANA

CATALOGO DE PATRONES

Diseño de software basado en Patrones

Edwin Avila Gómez
Vladimir Garcia Gutierrez

10/09/2010

Tabla de contenido

PATRON: ABSTRAC FACTORY

CATEGORIAS	Orientado a clases

DESCRIPCION

Proporcionar una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.

Aplicabilidad

- ✓ Cuando un sistema deba ser independiente de la creación, composición y representación de sus productos.
- ✓ Cuando un sistema deba ser configurado con una de las múltiples familias de productos.
- ✓ Cuando un conjunto de objetos relacionados se diseña para ser usado conjuntamente.
- ✓ Cuando se desea proporcionar una biblioteca de productos de los que sólo se quiere conocer su interfaz

VENTAJAS

- ✓ Aísla las clases concretas.
- ✓ Permite intercambiar fácilmente familias de Productos.
- ✓ Proporciona consistencia entre productos.

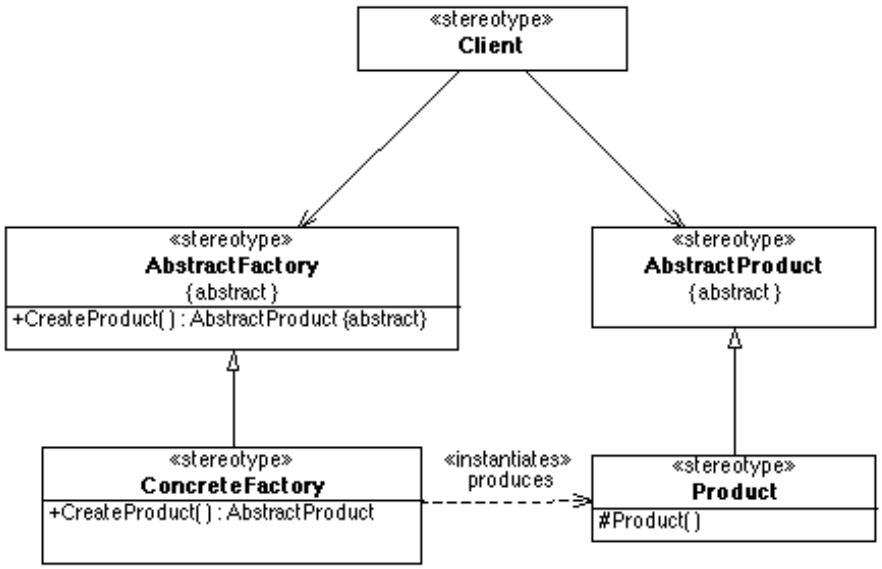
DESVENTAJAS

- ✓ Dificulta la incorporación de nuevas clases de productos

REFERENCIAS

http://es.wikipedia.org/wiki/Abstract_Factory_(patr%C3%B3n_de_dise%C3%B1o)
http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Abstract-factory.html

ESTRUCTURA



PATRON: Builder

CATEGORIAS	La diferencia reside en que *Constructor* se centra en la creación de los objetos paso a paso (parte a parte),

DESCRIPCION

Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.

Permitir la creación de una variedad de objetos complejos desde un objeto fuente (Producto), el objeto fuente se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas a interfaces comunes de la clase Abstract Builder.

VENTAJAS

- ✓ Reduce el acoplamiento.
- ✓ Permite variar la representación interna de estructuras compleja, respetando la interfaz común de la clase Builder.
- ✓ Se independiza el código de construcción de la representación. Las clases concretas que tratan las representaciones internas no forman parte de la interfaz del Builder.
- ✓ Cada ConcreteBuilder tiene el código especifico para crear y modificar una estructura interna concreta. Distintos Director con distintas utilidades (visores, parsers, etc) pueden utilizar el mismo ConcreteBuilder.
- ✓ Permita un mayor control en el proceso de creación del objeto. El Director controla la creación paso a paso, solo cuando el Builder ha terminado de construir el objeto lo recupera el Director.

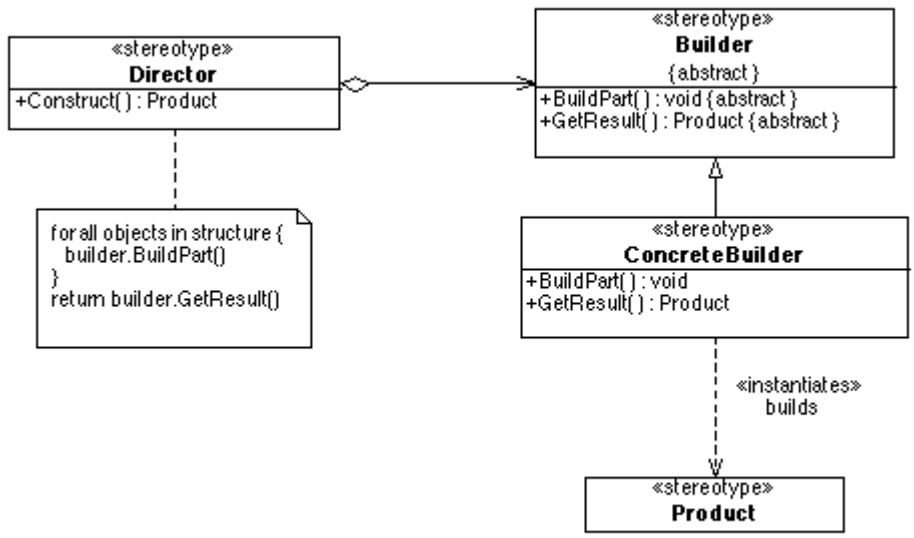
DESVENTAJAS

- ✓ Extremadamente complejo y costoso de implementar
- ✓ Incrementa el "heap" de memoria en procesos de construcción
- ✓ Lentifica el tiempo de construcción al hacer participar más clases en el proceso de instanciación

REFERENCIAS

http://www.dofactory.com/Patterns/PatternBuilder.aspx
http://es.wikipedia.org/wiki/Builder_(patr%C3%B3n_de_dise%C3%B1o)

ESTRUCTURA



PATRON: Factory Method

CATEGORIAS	cualquier sistema que requieren abstracción para la creación de objetos

DESCRIPCION

Consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. Es una simplificación del Abstract Factory, en la que la clase abstracta tiene métodos concretos que usan algunos de los abstractos; según usemos una u otra hija de esta clase abstracta, tendremos uno u otro comportamiento

VENTAJAS

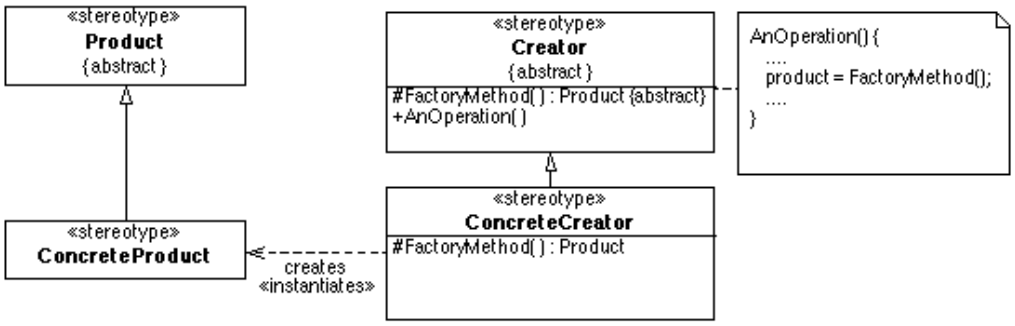
se unen jerarquías de clase paralelas proporcionando relacioes para la creación de subclase y la extensión

DESVENTAJAS

no saben de antemano todas las clases que deben ser creadas aplazando la creación de objetos a las clases con el contexto, la hoja de necesita para simular la construcción de objetos virtuales

http://www.vico.org/pages/PatronsDisseny/Pattern%20Factory%20Method/index.html
http://es.wikipedia.org/wiki/Factory_Method_(patr%C3%B3n_de_dise%C3%B1o)
http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Factory-method.html

ESTRUCTURA



PATRON: PROTOTYPE

CATEGORIAS

DESCRIPCION

tiene como finalidad crear nuevos objetos duplicándolos, clonando una instancia creada previamente

Este patrón es motivo donde en ciertos escenarios es preciso abstraer la lógica que decide que tipos de objetos utilizará una aplicación, de la lógica que luego usarán esos objetos en su ejecución. Los motivos de esta separación pueden ser variados, por ejemplo, puede ser que la aplicación deba basarse en alguna configuración o parámetro en tiempo de ejecución para decidir el tipo de objetos que se debe crear

Cuando en un sistema se usen este patrón en muchos objetos de negocio, conviene tener un prototype manager para preguntarle si tal o cual objeto es prototípico.

VENTAJAS

aísla clases concretas hace familias de producto de cambio permisos fáciles que añaden y productos de quitar en permisos de tiempo de ejecución que especifican nuevos objetos por cambiando structurepermits la especificación de nuevos objetos por cambiando valores promueven la consistencia entre productos reduce la subclasificación

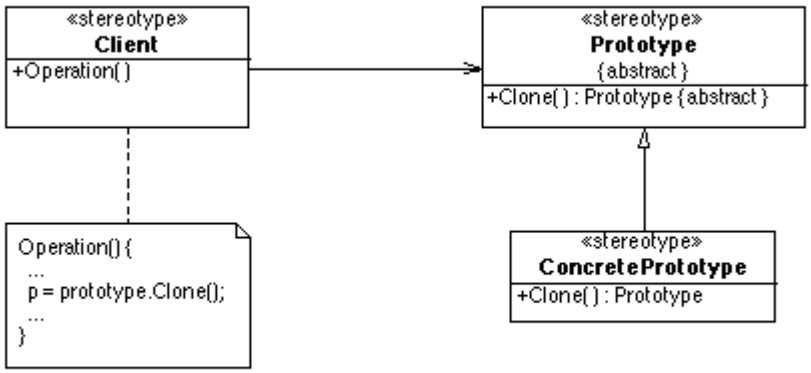
DESVENTAJAS

necesite una familia de objetos de producto relacionados de ser usado juntos y si necesita la ejecución de esta coacción, necesita un sistema para apoyar múltiples familias de interfaz de necesidad de clases para crear objetos sin saber sus tipos

REFERENCIAS

http://www.vico.org/pages/PatronsDisseny/Pattern%20Prototype/
http://es.wikipedia.org/wiki/Prototype_(patrón_de_diseño)
http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Prototype.html

ESTRUCTURA



PATRON: SINGLETON

CATEGORIAS

CREACIONAL *Relacionada con Fábrica Abstracta*, *Constructor* y *Prototipo*.

DESCRIPCION

Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación

VENTAJAS

- ✓ Proporciona el control para singularizar o en múltiples casos reducen el espacio de nombres global.
- ✓ El acceso a la "InstanciaÚnica" está más controlado.
- ✓ Se reduce el espacio de nombres (frente al uso de variables globales)
- ✓ Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario".
- ✓ Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).
- ✓ Es más flexible que la alternativa de las "operaciones de clase", además de que éstas (en C++), al ser funciones miembro estáticas, no son virtuales, luego no pueden ser especializadas mediante herencia ni redefinidas polimórficamente

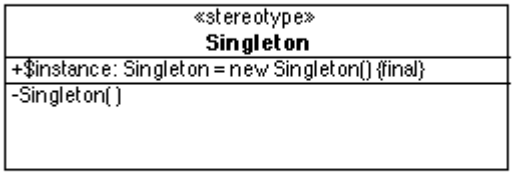
DESVENTAJAS

- ✓ debe controlar el número de los casos de una clase
- ✓ En la versión estática una vez que el/los objetos están instanciados, no se eliminan nunca de memoria. La garbage colector no podrá alcanzarlos ya que están referenciados por la clase y esta no los desreferenciará nunca hasta que el sistema se resetee.
- ✓ Si bien el patrón permite la herencia de los objetos singleton, la utilización de la misma debe ser controlada y planificada.
- ✓ La versión registro obliga a tener un objeto más en memoria.
- ✓ En programas multi-threads su utilización puede ser delicada, si al mismo tiempo dos hilos intentan instanciar un objeto a la vez, solo uno de ellos podría hacerlo.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Singleton.html
http://www.vico.org/pages/PatronsDisseny/Pattern%20Singleton/

ESTRUCTURA



PATRON: ADAPTER

CATEGORIAS	PATRON ESTRUCTURAL

DESCRIPCION

1) Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

El patrón **Adapter** (Adaptador) se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda.

Convierte la interfaz de una clase en otra interfaz que el cliente espera. *Adapter* permite a las clases trabajar juntas, lo que de otra manera no podría hacerlo debido a sus interfaces incompatibles

VENTAJAS

- ✓ Permite que un único *Adapter* trabaje con muchos *Adaptees*, es decir, el *Adapter* por sí mismo y las subclases (si es que la tiene).
- ✓ El *Adapter* también puede agregar funcionalidad a todos los *Adaptees* de una sola vez.
- ✓ Incrementa el rehuso de de clases por del desacomplamineto de la interfaz desde la implementación.

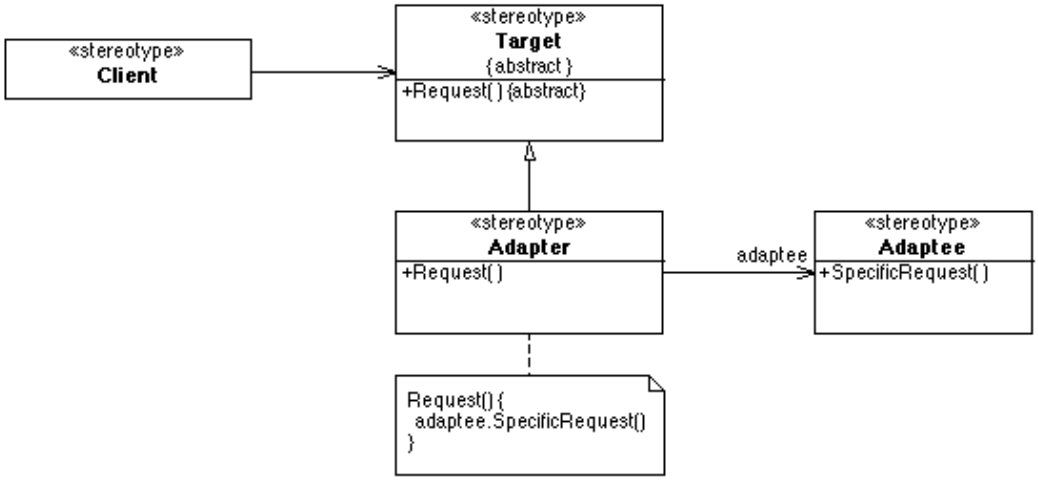
DESVENTAJAS

- ✓ Adapta Adaptee a Target encargando a una clase Adaptee concreta. Como consecuencia, una clase adaptadora no funcionará cuando se desea adaptar una clase y todas sus subclases
- ✓ Hace difícil sobrescribir el comportamiento de Adaptee. Esto requerirá derivar Adaptee y hacer que Adapter se refiera a la subclase en lugar que al Adaptee por sí mismo
- ✓ A nivel de clases Inflexibilidad, puesto que un sólo adaptador no puede trabajar con una clase y sus hijos a la vez.
- ✓ A nivel de objetos dificultad para redefinir el comportamiento de la clase adaptada.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Adapter.html
http://es.wikipedia.org/wiki/Adapter_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Adapter%20Class/

ESTRUCTURA



PATRON: Bridge

CATEGORIAS	PATRON ESTRUCTURAL, Estructura, a nivel de objetos

DESCRIPCION

Separar una abstracción de su implementación para permitir que ambos puedan variar independientemente.

El patrón Bridge, también conocido como Handle/Body, es una técnica usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra.

Esto es, se desacopla una abstracción de su implementación para que puedan variar independientemente.

VENTAJAS

- ✓ La separación entre interfaz e implementación permite configurar (y cambiar) esta relación dinámicamente, en tiempo de ejecución (y no de compilación) Además, esta independencia favorece una mejor estructuración en niveles del sistema.
- ✓ La citada independencia de variación repercute en una mayor extensibilidad.
- ✓ Se favorece el encapsulamiento, al ocultarse a los clientes detalles referentes a la implementación, como la compartición de implementaciones,

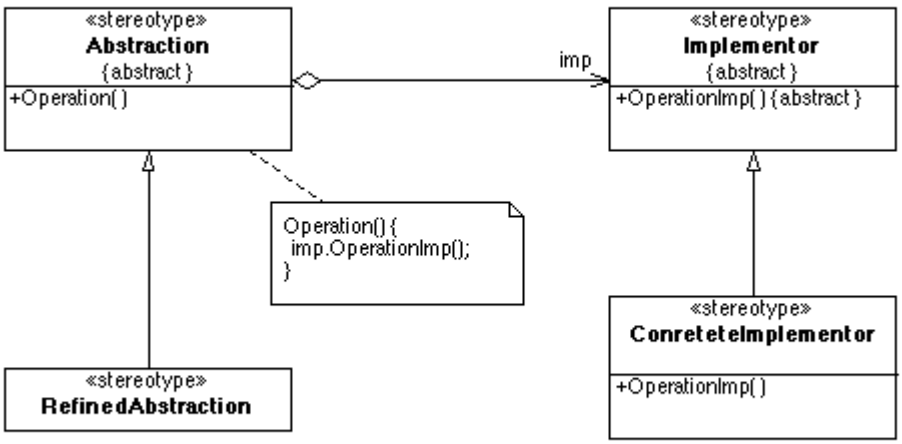
DESVENTAJAS

necesite una clase que es reutilizable en contextos múltiples, imprevistos tiene que de acoplar una clase y su puesta en práctica

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Bridge.html
http://es.wikipedia.org/wiki/Bridge_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Bridge/

ESTRUCTURA



PATRON: Composite

CATEGORIAS	**PATRON ESTRUTURAL,** Estructura, a nivel de objetos

DESCRIPCION

Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

El patrón **Composite** sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva y a una estructura en forma de árbol.

Esto simplifica el tratamiento de los objetos creados, ya que al poseer todos ellos una interfaz común, se tratan todos de la misma manera.

Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos

VENTAJAS

- ✓ Se crea una jerarquía de objetos básicos y de composiciones de estos
- ✓ Simplifica el cliente al tratar todos de igual forma
- ✓ Facilita agrega nuevas clases insertándolas en un contenedor compatible.
- ✓ Generaliza el diseño.

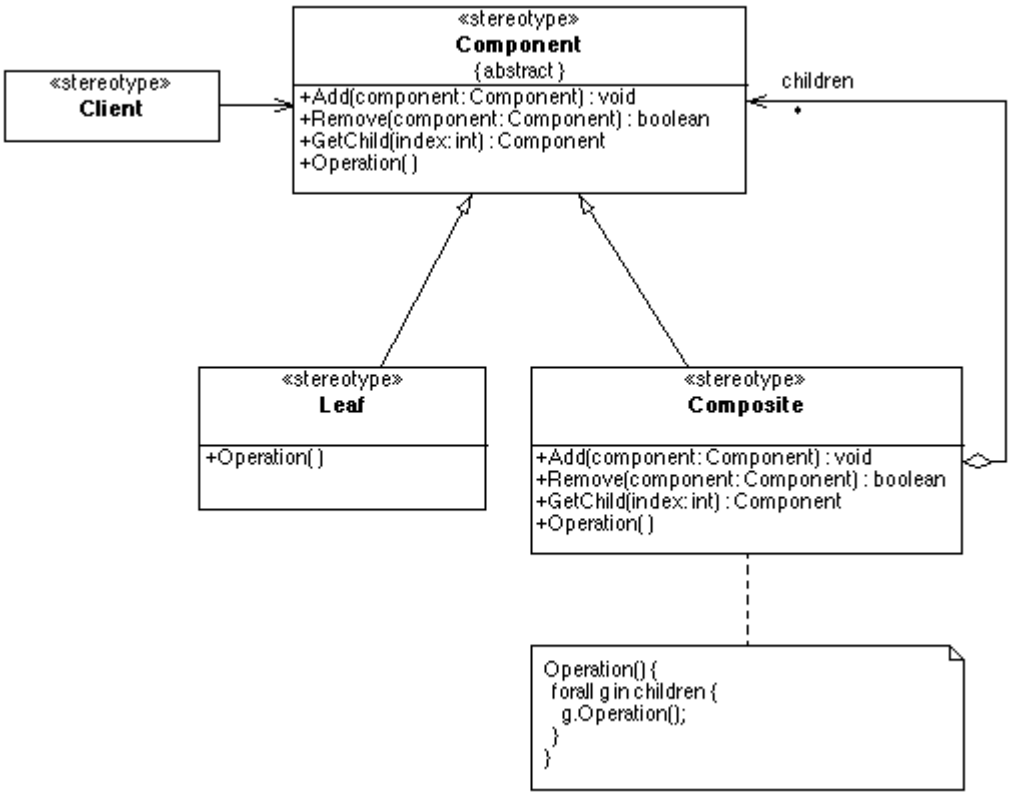
DESVENTAJAS

la necesidad de proteger a clientes de saber si un objeto es un todo o una parte

REFERENCIAS

http://www.vico.org/pages/PatronsDisseny/Pattern%20Composite/
http://es.wikipedia.org/wiki/Composite_(patrón_de_diseño)
http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Composite.html

ESTRUCTURA



PATRON: Decorator

CATEGORIAS	PATRON ESTRUCTURAL, Estructura, a nivel de objetos.

DESCRIPCION

Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

El patrón **Decorator** responde a la necesidad de añadir dinámicamente funcionalidad a un Objeto. Esto nos permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.

Se crea a partir de *Ventana* la subclase abstracta *VentanaDecorator* y, heredando de ella, *BordeDecorator* y *BotonDeAyudaDecorator*. *VentanaDecorator* encapsula el comportamiento de *Ventana* y utiliza composición recursiva para que sea posible añadir tantas "capas" de Decorators como se desee. Podemos crear tantos Decorators como queramos heredando de *VentanaDecorator*

VENTAJAS

- ✓ Aporta una mayor flexibilidad que la herencia estática, permitiendo, entre otras cosas, añadir una funcionalidad dos o más veces.
- ✓ Evita concentrar en lo alto de la jerarquía clases "guiadas por las responsabilidades", es decir, que pretenden (en vano) satisfacer todas las posibilidades.
- ✓ De esta forma las nuevas funcionalidades se componen de piezas simples que se crean y se combinan con facilidad, independientemente de los objetos cuyo comportamiento extienden.

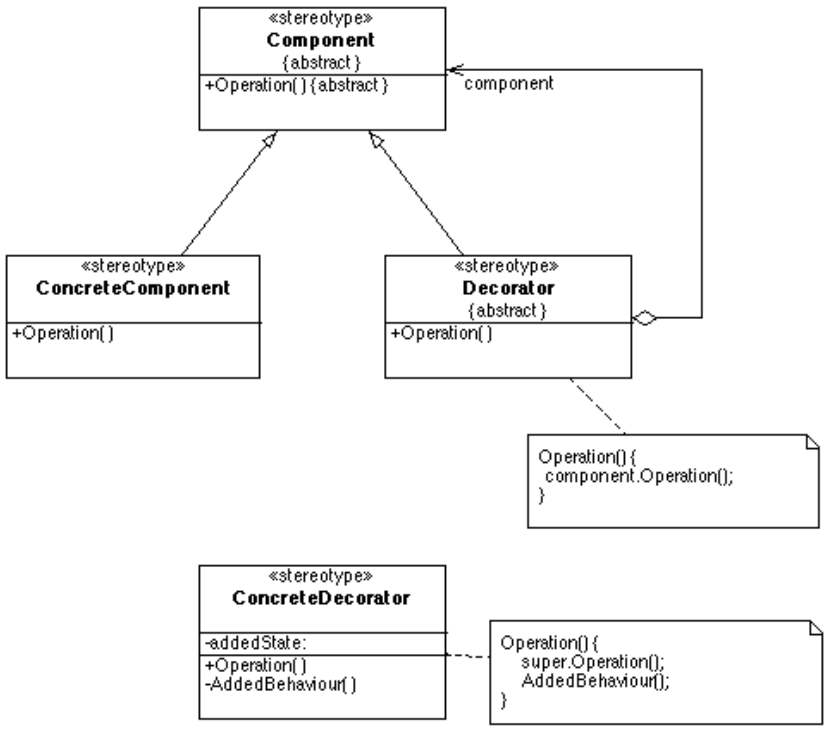
DESVENTAJAS

- ✓ La transparencia tiene el inconveniente de que, a pesar de ser diferentes por muchos motivos, un decorador y un componente son indistinguibles, por lo que no se puede confiar en la identidad de los objetos.
- ✓ Gran fragmentación; el sistema se llena de gran cantidad de objetos pequeños, lo cual puede dificultar el aprendizaje de su funcionamiento y su depuración, aunque el que lo domine puede adaptarlo fácilmente

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Decorator.html
http://es.wikipedia.org/wiki/Decorator_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Decorator/

ESTRUCTURA



PATRON: Facade

CATEGORIAS	Estructura, a nivel de objetos.

DESCRIPCION

Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.

Sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.

Describe como representar subsistemas completos como objetos.

Facade puede:

- Hacer una biblioteca de software más fácil de usar y entender, ya que facade implementa métodos convenientes para tareas comunes.
- Hacer el código que usa la librería más legible, por la misma razón.
- Reducir la dependencia de código externo en los trabajos internos de una librería, ya que la mayoría del código lo usa Facade, permitiendo así más flexibilidad en el desarrollo de sistemas.
- Envolver una colección mal diseñada de APIs con un solo API bien diseñado. lu

VENTAJAS

- ✓ puede reducir las dependencias de compilación,
- ✓ permite a los clientes de varias habilidades para acceder a la capa adecuada del sistema,
- ✓ promueve la estratificación de los sistemas, proporciona bajo acoplamiento entre los clientes y los subsistemas

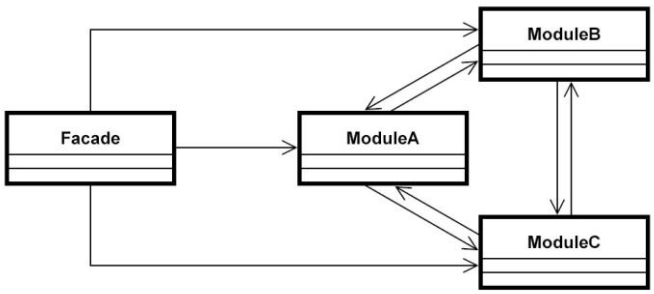
DESVENTAJAS

- ✓ necesidad de subsistemas de capa,
- ✓ necesidad de proporcionar abstracción entre subsistema y el cliente

REFERENCIAS

http://es.wikipedia.org/wiki/Facade_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Facade/
http://www.dofactory.com/Patterns/PatternFacade.aspx

ESTRUCTURA



PATRON: Flyweight

CATEGORIAS	Estructura, a nivel de objetos.

DESCRIPCION

Mejorar la eficiencia a la hora de mantener una gran cantidad de objetos “de grano fino”, usando la idea de la compartición. Un “Peso Mosca” es un objeto compartido que puede ser usado en diferentes contextos simultáneamente, de forma transparente, Indistinguible.

El concepto clave aquí es la distinción entre el ESTADO INTRÍNSECO (información independiente del contexto del objeto, almacenada en el mismo y que favorece su compartición) y el ESTADO EXTRÍNSECO (información que depende y varía con el contexto donde se use el objeto, no puede ser compartida y se la proporciona el cliente que lo usa, a través de la fábrica) del objeto compartido

VENTAJAS

- ✓ Mejora en la eficiencia, sobre todo en relación con el ahorro en el almacenamiento, que aumenta paralelamente con la compartición de objetos y es función de :

(a) la reducción en el nº total de instancias.
(b) la cantidad de estado intrínseco por objeto.
(c) si el estado extrínseco es computado (calculado) o almacenado.

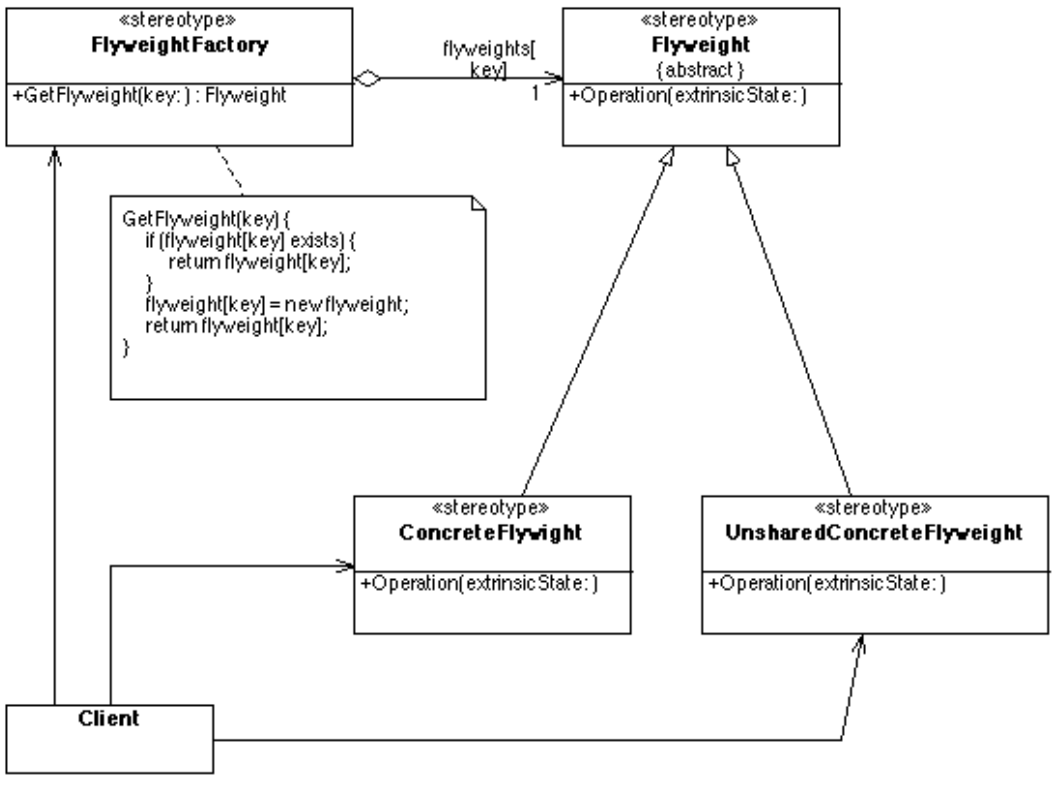
DESVENTAJAS

Se introducen costes en tiempo de ejecución asociados a las transferencias, búsquedas y/o computación del estado extrínseco (y su almacenamiento).

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Flyweight.html
http://www.dofactory.com/Patterns/PatternFlyweight.aspx
http://es.wikipedia.org/wiki/Flyweight_(patrón_de_diseño)

ESTRUCTURA



PATRON: PROXY

CATEGORIAS	Estructura, a nivel de objetos

DESCRIPCION

Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

El patrón **Proxy** (Representante) se utiliza como intermediario para acceder a un objeto, permitiendo controlar el acceso a él.

Necesitamos crear objetos que consumen muchos recursos, pero no queremos instanciarlos a no ser que el cliente lo solicite o se cumplan otras condiciones determinadas.

Tipos:

- **Remote**
- **Virtual**
- **Copy-on-write**
- **Protection**
- **Cache**
- **Firewall**
- **Syncronice**
- **Smartreference**

VENTAJAS

- ✓ mantenimiento mayor y la extensibilidad del sistema.
- ✓ puede reducir las dependencias de compilación.
- ✓ promueve la estratificación de los sistemas.
- ✓ proporciona acoplamiento débil entre los clientes y los subsistemas
- ✓ Se introduce un nivel de indirección en el acceso al objeto, que permite al *apoderado remoto* ocultar el hecho de que el objeto reside en un espacio de direcciones distinto, al *apoderado virtual* realizar optimizaciones como la creación de objetos por demanda y al *apoderado de protección* y a las *referencias "inteligentes"* realizar tareas adicionales de vigilancia sobre el objeto al que se accede. No obstante todo esto *puede resultar un inconveniente*, por motivos de claridad e inteligibilidad del diseño.
- ✓ Facilita otra optimización, relacionada con la creación de objetos por demanda : la técnica de *COPY-ON-WRITE*, que sólo hace efectiva la copia de un objeto oneroso
  cuando el acceso a él es de escritura, no de lectura

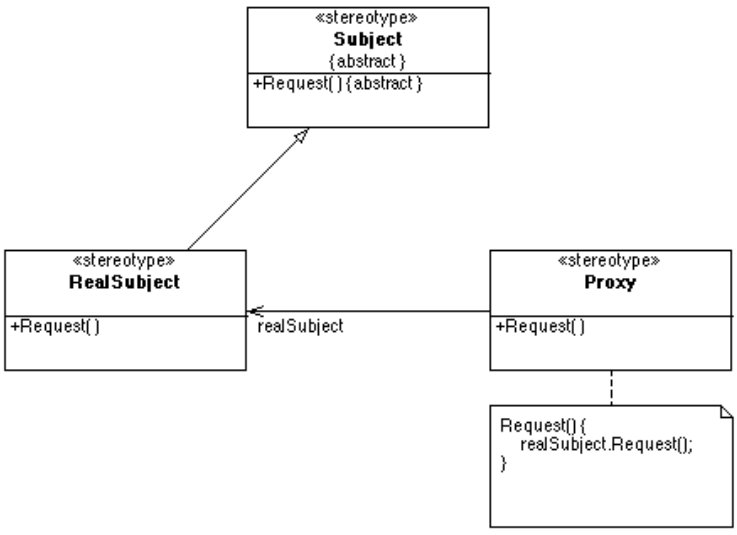
DESVENTAJAS

- ✓ Tiene que controlar el acceso a otro objeto,
- ✓ Tiene que proporcionar la abstracción entre el subsistema y el cliente

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Proxy.html
http://www.dofactory.com/Patterns/PatternProxy.aspx
http://es.wikipedia.org/wiki/Proxy_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Proxy/

ESTRUCTURA



PATRON: Chain of responsibility

CATEGORIAS	Comportamiento, a nivel de objetos

DESCRIPCION

Proporcionar a más de un objeto la capacidad de atender una petición, para así evitar el acoplamiento con el que objeto que hace la petición. Se forma con estos objetos una cadena, en la cual cada objeto o satisface la petición o la pasa al siguiente

Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

Permite establecer una cadena de objetos receptores a través de los cuales se pasa una petición formulada por un objeto emisor. Cualquiera de los objetos receptores puede responder a la petición en función de un criterio establecido.

VENTAJAS

- ✓ Se reduce el acoplamiento, puesto que se libera al objeto que hace la petición de conocer quien se la atiende. No sólo eso, sino que, además, los miembros de la cadena no conocen la estructura entera, sino sólo su sucesor en la misma.
- ✓ Mayor flexibilidad, puesto que se puede añadir o modificar la capacidad de atender una petición, simplemente haciendo cambios en la cadena de responsabilidades, dinámicamente (en tiempo de ejecución).

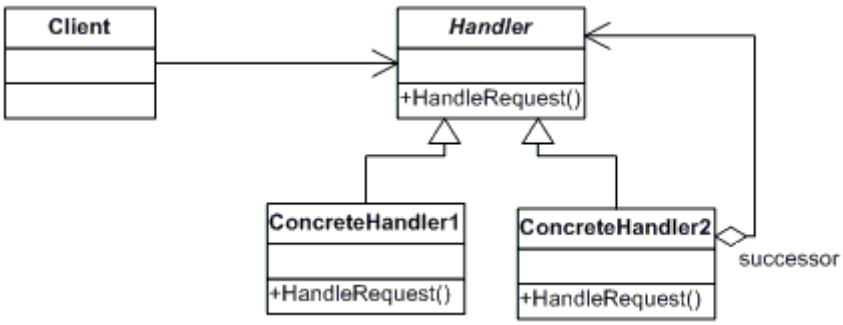
DESVENTAJAS

- ✓ No se garantiza la recepción y satisfacción de una petición, pues puede haber errores en la configuración de la cadena o también puede ocurrir que ningún elemento de la cadena sepa atender la petición.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Chain-of-responsibility.html
http://es.wikipedia.org/wiki/Chain_of_Responsibility_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Chain%20of%20Responsability/
http://www.dofactory.com/Patterns/PatternChain.aspx

ESTRUCTURA



PATRON: Command

CATEGORIAS	Comportamiento, a nivel de objetos

DESCRIPCION

Problema
Uno de los elementos principales de una interfaz gráfica de usuario (GUI) son los menús
desplegables. En el interior de un menú se encuentran una serie de operaciones,
funciones o facilidades que una aplicación ofrece con relación a un tema en concreto
(Archivo, Edición, Ver, Insertar, Formato, etc...). Las distintas entradas de un menú tienen
la característica de que al ser "pinchadas" con el ratón o al ser seleccionadas mediante
teclado ejecutan una operación concreta

VENTAJAS

- ✓ incrementa la reutilización de las clases por la disociación de la interfaz de la aplicación
- ✓ prevé acoplamiento débil entre los clientes y los subsistemas.
- ✓ le permite variar el comportamiento de una clase independiente de su contexto

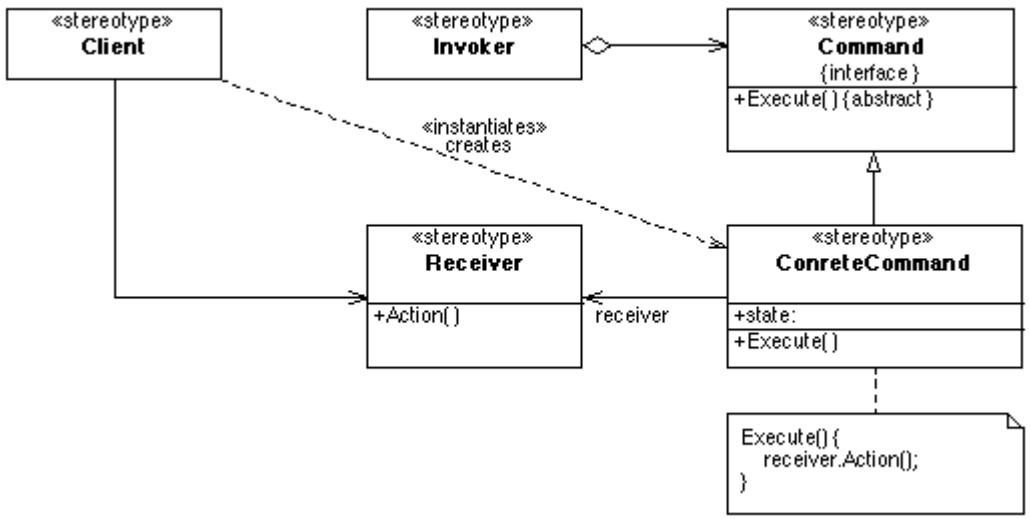
DESVENTAJAS

- ✓ necesidad de modificar las conductas objeto de forma dinámica sin afectar a los objetos existentes,
- ✓ necesidad de hacer cola las solicitudes de ejecutar en el futuro - como deshacer

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Command.html
http://www.vico.org/pages/PatronsDisseny/Pattern%20Command/
http://es.wikipedia.org/wiki/Command_(patrón_de_diseño)
http://www.dofactory.com/Patterns/PatternCommand.aspx

ESTRUCTURA



PATRON: Interpreter

CATEGORIAS	Comportamiento, a nivel de clases

DESCRIPCION

Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.

Dado un lenguaje, define una representación para su gramática junto con un intérprete del lenguaje.

Se usa para definir un lenguaje para representar expresiones regulares que representen cadenas a buscar dentro de otras cadenas. Además, en general, para definir un lenguaje que permita representar las distintas instancias de una familia de problemas.

VENTAJAS

- ✓ Facilidad para cambiar o extender la gramática, mediante herencia, dado que las diferentes reglas de la gramática se representan con clases.
- ✓ Facilidad para implementar la gramática, dado que las implementaciones de las clases nodo del árbol sintáctico son similares, pudiendo usarse para ello generadores automáticos de código.
- ✓ Facilidad para introducir nuevas formas de "interpretar" las expresiones en la gramática (tener en cuenta el patrón *Visitante*).

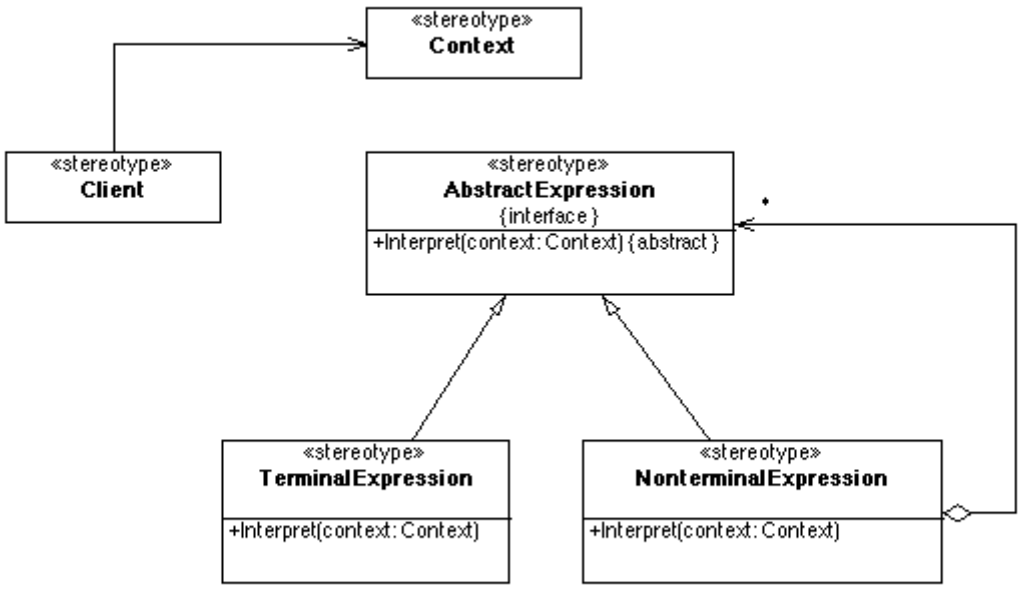
DESVENTAJAS

Limitación en el tipo de gramáticas (y, por extensión, de problemas) para los que sirve esta aproximación: simples y sin grandes necesidades de eficiencia. Aunque el uso de otros patrones puede mitigar este problema, hay circunstancias en que un *Intérprete* no es lo más apropiado.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Interpreter.html
http://es.wikipedia.org/wiki/Interpreter_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Interpreter/
http://www.dofactory.com/Patterns/PatternInterpreter.aspx

ESTRUCTURA



PATRON: Iterator

CATEGORIAS	Comportamiento, a nivel de objetos

DESCRIPCION

Proporcionar una forma de acceder secuencialmente a los elementos de un objeto compuesto por agregación sin necesidad de desvelar su representación interna

En diseño de software, el patrón de diseño **Iterador**, define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección. Algunos de los métodos que podemos definir en la interfaz Iterador son:

Primero(), Siguiente(), HayMas() y ElementoActual()

Este patrón de diseño permite recorrer una estructura de datos sin que sea necesario conocer la estructura interna de la misma.

VENTAJAS

- ✓ Se incrementa la flexibilidad, dado que para permitir nuevas formas de recorrer una estructura basta con modificar el iterador en uso, cambiarlo por otro (¡parametrización!) o definir uno nuevo.
- ✓ Se enfatiza la distribución de responsabilidades (y, por ello, la simplicidad de los elementos del sistema), dado que la interfaz del agregado se ve libre de una serie de operaciones, más propias de la interfaz del iterador.
- ✓ Se facilitan el paralelismo y la concurrencia, puesto que, como cada iterador tiene consciencia de su estado en cada momento, es posible que dos o más iteradores recorran una misma estructura simultánea o solapadamente

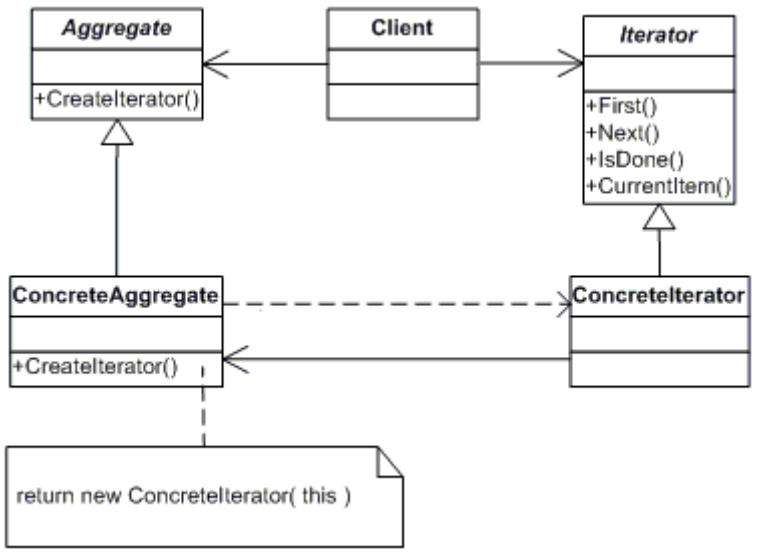
DESVENTAJAS

- ✓ La necesidad de proteger a los clientes de saber si un objeto es un todo o una parte

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Iterator.html
http://es.wikipedia.org/wiki/Iterador_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Iterator/

ESTRUCTURA



PATRON: Mediator

CATEGORIAS	Comportamiento, a nivel de objetos.

DESCRIPCION

Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

Coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.

Define un objeto que encapsula como interactúa un conjunto de objetos

VENTAJAS

- ✓ Se limita el uso de la herencia, al concentrar en el "Mediador" un comportamiento que, de otro modo, estaría distribuido entre diversos objetos. Modificar este comportamiento puede requerir especializar el "Mediador", pero ya no afectará a los objetos que interactúan.
- ✓ Al reducir el acoplamiento entre los objetos que interactúan es más fácil variar o reutilizar dichos objetos y el "Mediador", independientemente.
- ✓ Se simplifican los protocolos de comunicación entre los objetos, al sustituir las relaciones "muchos-a-muchos" por relaciones "uno-a-muchos", que son más sencillas de entender, mantener y extender.
- ✓ Al hacer de la mediación un concepto independiente, encapsulado en un objeto aparte, se favorece el centrar la atención en la interacción entre objetos, al margen del
- ✓ comportamiento individual de cada uno.

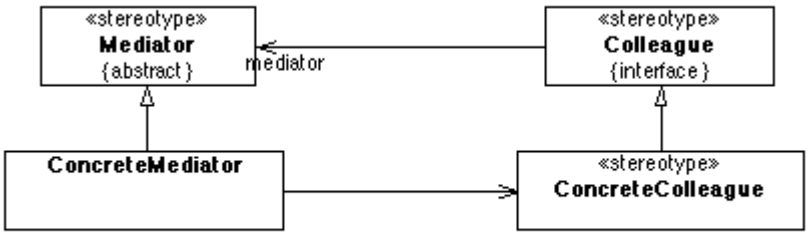
DESVENTAJAS

Se cambia complejidad en la interacción por complejidad en el "Mediador", que se
puede convertir en el elemento más complejo del conjunto de objetos. Este carácter
monolítico lo hará difícil de mantener11

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Mediator.html
http://es.wikipedia.org/wiki/Mediator_(patrón_de_diseño)

ESTRUCTURA



PATRON: Memento

CATEGORIAS	Comportamiento, a nivel de objetos.

DESCRIPCION

Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.
Tiene como finalidad almacenar el estado de un objeto (o del sistema completo) en un momento dado de manera que se pueda restaurar en ese punto de manera sencilla. Para ello se mantiene almacenado el estado del objeto para un instante de tiempo en una clase independiente de aquella a la que pertenece el objeto (pero sin romper la encapsulación), de forma que ese recuerdo permita que el objeto sea modificado y pueda volver a su estado anterior.
Se usa este patrón cuando se quiere poder restaurar el sistema desde estados pasados y por otra parte, es usado cuando se desea facilitar el hacer y deshacer de determinadas operaciones, para lo que habrá que guardar los estados anteriores de los objetos sobre los que se opere (o bien recordar los cambios de forma incremental).

VENTAJAS

- ✓ Se salvaguarda el encapsulamiento, impidiendo la exposición pública de información que sólo el "Originador" debe manejar, pero que, por otra parte, puede (y
- ✓ debe) almacenarse fuera del mismo.
- ✓ Se simplifica al objeto "Originador", que en otros diseños posibles, que también
- ✓ preservarían el encapsulamiento, tendría que hacerse cargo del mantenimiento de todas
- ✓ las versiones del estado interno en que los clientes estuvieran interesados.

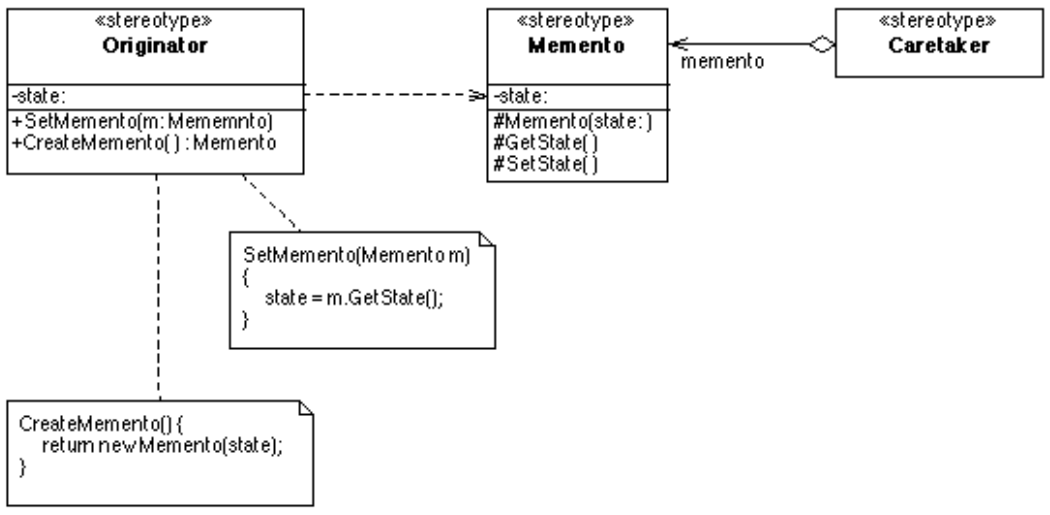
DESVENTAJAS

- ✓ El uso de "Recuerdos" puede convertirse en inapropiado, por el excesivo consumo de recursos, en el caso de que la información que el "Originador" deba almacenar en el "Recuerdo" sea considerable o las peticiones de los clientes solicitando "Recuerdos" sea demasiado frecuente.
- ✓ Puede resultar problemático en ciertos lenguajes el mantenimiento de dos interfaces distintas ("amplia" para el "Originador" y "estrecha" para el "Cuidador") para acceder al "Recuerdo".
- ✓ Puede haber costes ocultos en el cuidado de los "Recuerdos", pues los "Cuidadores" son los responsables de eliminar los "Recuerdos" que vigilan, pese a no conocer sus interioridades, lo cual dificulta su labor.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Memento.html
http://www.vico.org/pages/PatronsDisseny/Pattern%20Memento/
http://es.wikipedia.org/wiki/Memento_(patrón_de_diseño)
http://www.dofactory.com/Patterns/PatternMemento.aspx

ESTRUCTURA



PATRON: Observer

CATEGORIAS	Comportamiento a nivel de objetos.

DESCRIPCION

Un efecto lateral muy frecuente en aquellos sistemas que se fragmentan en un conjunto de clases que cooperan es la necesidad de mantener la consistencia entre los distintos objetos interrelacionados. Para no recurrir a soluciones fuertemente acopladas (que reducen la posibilidad de reutilización), este patrón define una dependencia "uno-amuchos" entre objetos, para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente.

VENTAJAS

- ✓ Dado que el "Asunto" no conoce la clase de "Observadores" que tiene, el acoplamiento que existe es mínimo y abstracto. Por ello, tanto unos como otros pueden variar y evolucionar en diferentes niveles de detalle de forma independiente.
- ✓ Se facilita y simplifica el "radiado" de las notificaciones. El "Asunto" sólo tiene que comunicar los cambios; luego, cada "Observador" es libre para hacer o no caso. Esto permite gran libertad a la hora de dar de alta o baja a los "Observadores".

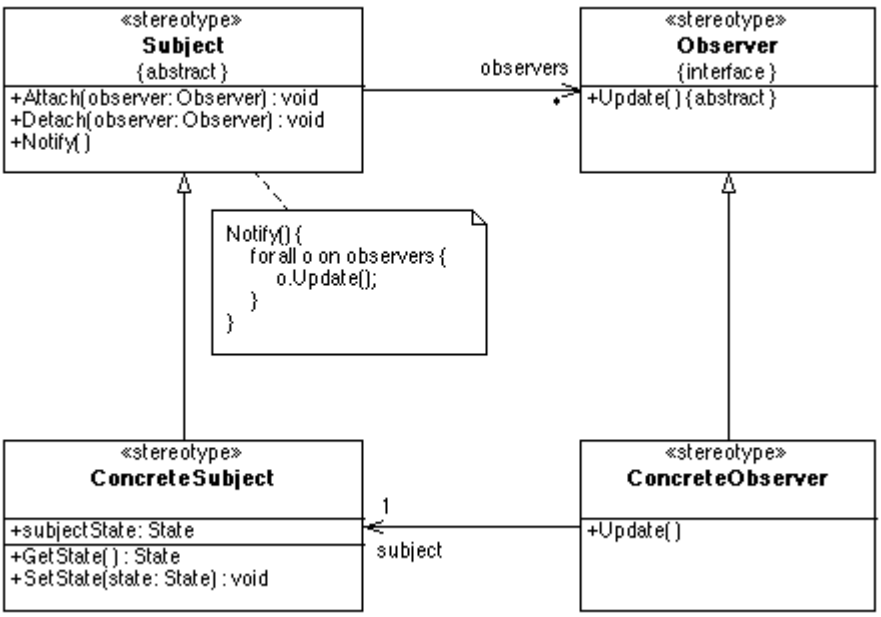
DESVENTAJAS

Dado el desconocimiento que unos "Observadores" tienen de los otros, cualquier operación aparentemente inofensiva puede originar una cascada de costosas actualizaciones. Además, sin un protocolo preciso, en el que se especifique lo que cambia, el esfuerzo se multiplica.

REFERENCIAS

http://www.tml.tkk.fi/~pnr/Tik-76.278/gof/html/Observer.html
http://es.wikipedia.org/wiki/Observer_(patrón_de_diseño)
http://www.vico.org/pages/PatronsDisseny/Pattern%20Observer/

ESTRUCTURA



PATRON: STATE

CATEGORIAS

Comportamiento a nivel de objetos

DESCRIPCION

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto.

Aplicabilidad

- ✓ El comportamiento de un objeto depende de su estado, y debe cambiar en tiempo de ejecución dependiendo de su estado.
- ✓ Las operaciones tienen largas sentencias convencionales con múltiples ramas que dependen del estado del objeto. Este estado se suele representar por uno o más constantes enumeradas. Muchas veces son varias las operaciones que contiene esta misma estructura condicional. El patrón State pone cada rama de la condición en una clase aparte. Esto nos permite tratar el estado del objeto como un objeto de pleno derecho que puede variar independientemente de otros objetos.

VENTAJAS

- ✓ Dado que el comportamiento específico de cada estado está autocontenido en cada estado concreto, existe una gran flexibilidad para añadir nuevos estados y transiciones, mediante la definición de nuevas subclases.
- ✓ A pesar de que la distribución del comportamiento ante diferentes situaciones entre diferentes objetos incrementa el nº de clases y reduce la compacidad, es la mejor solución cuando hay muchos estados, pues evita las definiciones multicondicionales y grandes, tan indeseables como los procedimientos gigantes lo son en el estilo imperativo.
- ✓ Se hacen más explícitas las transiciones entre estados que sí todo estuviera concentrado en una sola clase (esto hace más inteligible el diseño), impidiendo además los estados internos inconsistentes (desde el punto de vista del contexto, los cambios de estado son ATÓMICOS).
- ✓ Si los objetos que representan los estados no tienen variables de instancia (esto es, el estado está totalmente definido en su propio tipo) entonces diferentes contextos pueden compartir estados, como PesoMosca sin estado intrínseco, sólo comportamiento

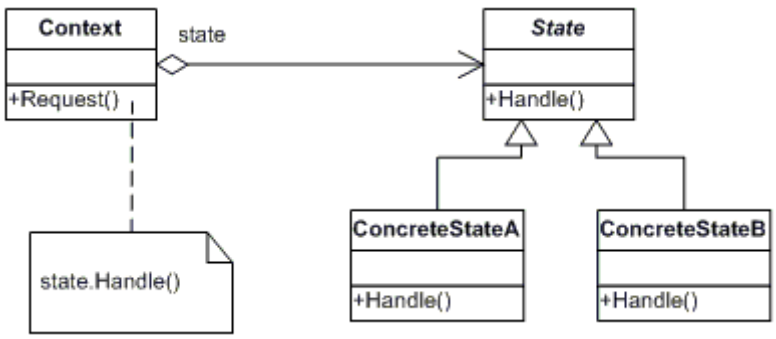
DESVENTAJAS

necesidad de romper muchos de los comportamientos con varias condiciones,
la necesidad de modificar las conductas objeto de forma dinámica sin afectar a los objetos existentes

REFERENCIAS

http://www.vico.org/pages/PatronsDisseny/Pattern%20State/
http://www.dofactory.com/Patterns/PatternState.aspx

ESTRUCTURA



PATRON: Strategy

CATEGORIAS	Comportamiento, a nivel de objetos.

DESCRIPCION

Definir una familia de algoritmos encapsulando por separado cada uno de ellos y haciéndolos, por tanto, intercambiables. Esto permite a los algoritmos variar con independencia de los clientes que los usan
Cuando usarlo

- ✓ Cuando muchas clases relacionadas sólo se diferencian en su comportamiento.
- ✓ Cuando se necesitan diferentes variantes de un algoritmo.
- ✓ Cuando un algoritmo usa datos que los clientes no tienen por qué conocer.
- ✓ Cuando una clase define muchos comportamientos, los cuales se manifiestan como definiciones condicionales múltiples de sus operaciones

VENTAJAS

- ✓ Las familias jerárquicas de estrategias definen algoritmos y comportamientos que enfatizan la reutilización. La herencia puede ayudar a sacar factor común a la funcionalidad de los algoritmos.
- ✓ · El encapsulamiento de algoritmos en clases separadas ofrece una ventajosa alternativa a la especialización por herencia del contexto para obtener un comportamiento diferente, que promueve la independencia, la facilidad de entender el diseño y la posibilidad de futuras extensiones.
- ✓ Se eliminan las costosas definiciones de comportamiento multicondicionales.
- ✓ Se posibilita ofrecer diferentes implementaciones del mismo comportamiento, en función de restricciones como el espacio en memoria o el tiempo de respuesta

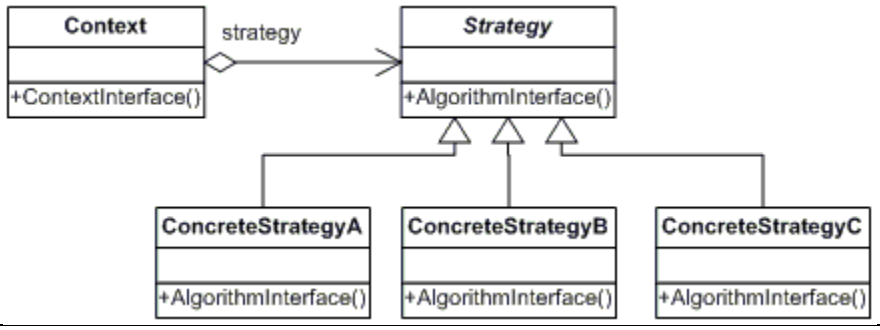
DESVENTAJAS

- ✓ Los clientes deben tener un cierto conocimiento de cada estrategia, para así poder elegir en cada situación cual es la más apropiada.
- ✓ Dado que todas las estrategias comportan una interfaz común, si las diferencias entre ellas es grande, es probable que mucha de la información que se les pasa no sea de utilidad más que a las más complejas.
- ✓ Puede producirse una gran explosión en el número de objetos del sistema. Esto se puede aliviar si las estrategias se implementan como objetos sin estado que los contextos pueden compartir

REFERENCIAS

http://www.vico.org/pages/PatronsDisseny/Pattern%20Strategy/
http://www.dofactory.com/Patterns/PatternStrategy.aspx

ESTRUCTURA



PATRON: Template Method

CATEGORIAS	Comportamiento, a nivel de clases.

DESCRIPCION

Definir el esqueleto de un algoritmo para una operación, dejando para sus subclases la capacidad de redefinir el funcionamiento de los pasos de este algoritmo, siempre y cuando la estructura del mismo permanezca intacta.

VENTAJAS

- ✓ Los "MetodosPlantilla" son una técnica fundamental para reutilizar código, especialmente en las librerías de clases, donde son la razón de ser de la "factorización de comportamiento común". Llevan a una estructura de control invertido, cuyo paradigma es el "Principio de Hollywood" : <<NO NOS LLAME, YA LE LLAMAREMOS>>, esto es, la clase padre invoca la implementación que la clase hija hace de operaciones primitivas, pero no al revés.

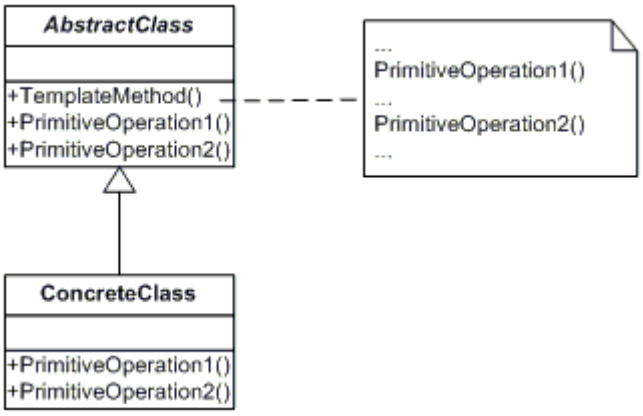
DESVENTAJAS

necesidad de flexibilidad en la creación de varios objetos compleja necesidad de muchos algoritmos con una variación

REFERENCIAS

http://www.dofactory.com/Patterns/PatternTemplate.aspx
http://www.vico.org/pages/PatronsDisseny/Pattern%20Template%20Method/

ESTRUCTURA



PATRON: Visitor

CATEGORIAS	Comportamiento, a nivel de objetos

DESCRIPCION

Representar una operación que está pensada para ser aplicada sobre los elementos de una estructura de objetos, permitiendo así definir y añadir un nuevo comportamiento sin necesidad de cambiar las clases de los elementos de la estructura de objetos.

VENTAJAS

- ✓ Es fácil añadir nuevas operaciones a la "Estructura De Objetos"; sólo hay que crear un nuevo "Visitante", en vez de cambiar todas las clases a las que queremos que afecte.
- ✓ Se juntan las operaciones relacionadas entre sí, separándose las que nada tienen que ver. Esto simplifica tanto las clases de los "Elementos" como los algoritmos definidos en los "Visitantes".
- ✓ Se pueden recorrer jerarquías formadas por objetos de distintos tipos (distinta clase padre), mientras que un Iterador no puede hacer esto.
- ✓ Se puede ir acumulando el estado a medida que se recorre la estructura, en vez de tener que pasarlo como parámetro o guardarlo en variables globales.

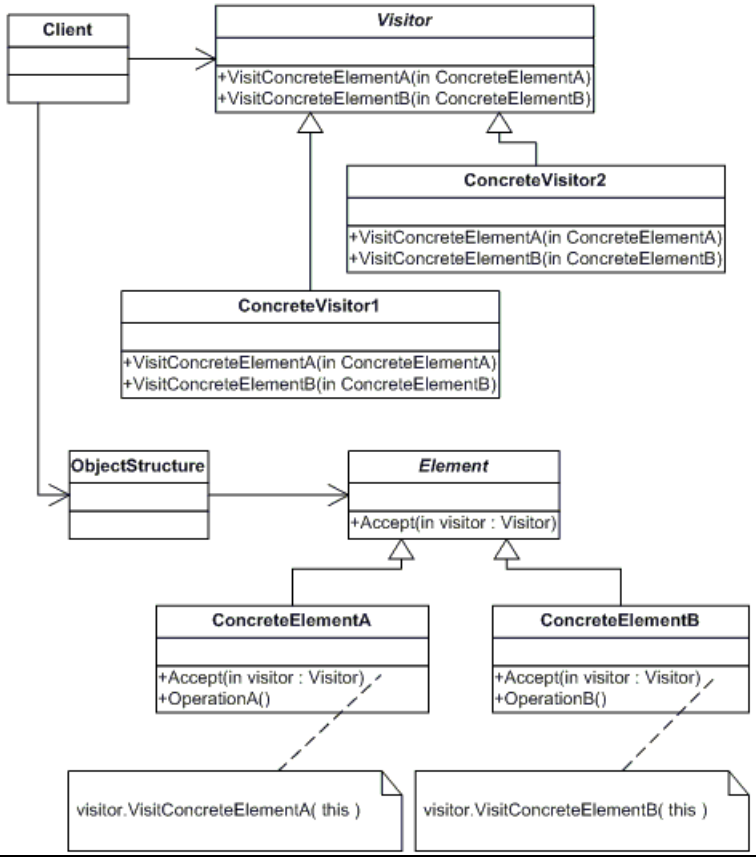
DESVENTAJAS

- ✓ Añadir un nuevo tipo de elemento a la estructura de objetos crea problemas, pues hay que tocar toda la estructura jerárquica de "Visitantes", lo cual es muy costoso.
- ✓ Frecuentemente se fuerza a los "Elementos" a proporcionar operaciones públicas que den acceso a su estado interno, con lo que se pone en peligro el encapsulamiento.

REFERENCIAS

http://www.dofactory.com/Patterns/PatternVisitor.aspx
http://www.vico.org/pages/PatronsDisseny/Pattern%20Visitor/

ESTRUCTURA



PATRONES DE ARQUITECTURA

Patrones de POSA

From Mud to Structure	Distributed Systems	Interactive Systems	Adaptable Systems
Layers Pipes and Filters Blackboard	Broker	Model-View-Controller Presentation-Abstraction-Control	Microkernel Reflection

Domain Logic Pattern	Mapping toRelational Databases	Web Presentation Patterns	Distribution Patterns	Offline Concurrency Patterns	Session State Patterns	Base Patterns
Transaction Script Domain Model Table Module Service Layer	**Data Source Architectural Patterns** Table Data Gateway Row Data Gateway Active Record Data Mapper **Object Relational Behavioral Patterns** Unit of Work Identity Map Lazy Load **Object Relational Structural Patterns** Identity Field Foreign Key Mapping Association Table Mapping Dependent Mapping Embedded Value Serialized LOB Single Table Inheritance Class Table Inheritance Table Inheritance Concrete Inheritance Inheritance Mappers **Object-Relational Metadata Mapping Patterns** Metadata Mapping Query Object Repository	Model View Controller Page Controller Front Controller Template View Transform View Two Step View Application Controller	Remote Façade Data Transfer Object	Optimistic Offline Lock Pesimistic Offline Lock Coarse-Grained Lock Implicit Lock	Client Session State Server Session State Database Session State	Gateway Mapper Layer Supertype Separated Interface Registry Value Object Money Special Case Plugin Service Stub Record Set

PATRON: Model View Controller Pattern (Patrón Modelo Viste Controlador – MVC-)

CATEGORIAS	Aplicaciones interactivas con una interfaz hombre-maquina flexible.

DESCRIPCION

El patrón arquitectural Modelo-Vista-Controlador (MVC) divide una aplicación interactiva en tres componentes. El modelo contiene la funcionalidad central y los datos. Las vistas exponen
información al usuario. Los controladores manejan las entradas del usuario. Un mecanismo de
change-propagation asegura la consistencia entre la interfaz de usuario y el modelo.

Usos conocidos: aplicaciones web, Smalltalk, MFC, ET++

VENTAJAS

Menor acoplamiento
1. Desacopla las vistas de los modelos.
2. Desacopla los modelos de la forma en que se muestran e ingresan los datos.

Mayor cohesión
1. Cada elemento del patrón esta altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).

Las vistas proveen mayor flexibilidad y agilidad
1. Se puede crear múltiples vistas de un modelo.
2. Se puede crear, añadir, modificar y elimina nuevas vistas dinámicamente.
3. Las vistas pueden anidarse.
4. Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
5. Se puede sincronizar las vistas.
6. Las vistas pueden concentrarse en diferentes aspectos del modelo.

Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales
1. Una vista que puede variar según sus capacidades.
2. Una vista para la Web y otras aplicaciones de escritorio.

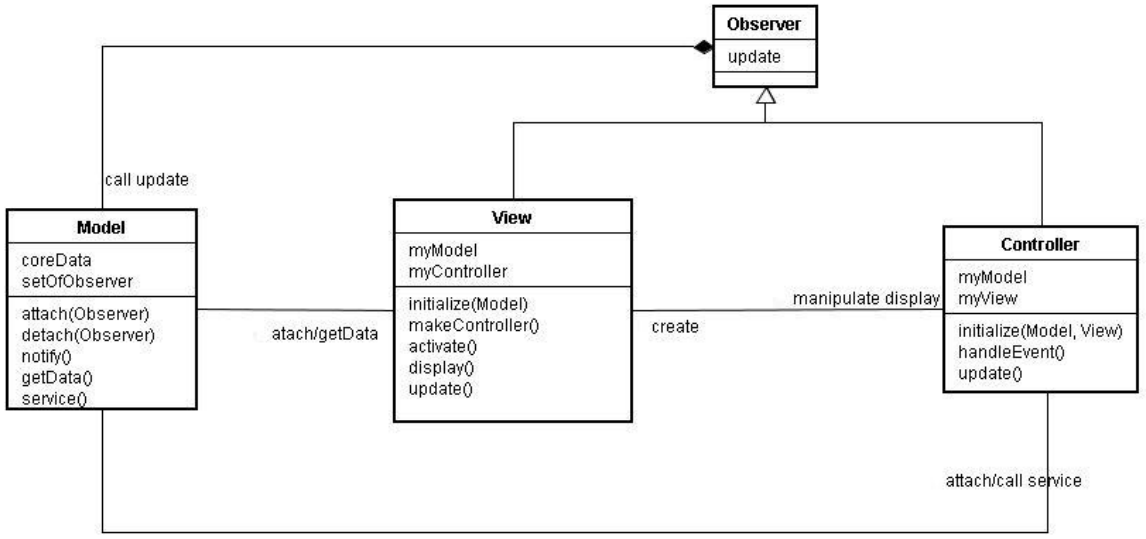
DESVENTAJAS

- Complejidad incrementada.
- Potencial para excesivo número de actualizaciones.
- Intima conexión entre vista y controlador.
- Estrecho acoplamiento de vistas y controladores a un modelo.
- Ineficiencia del acceso a los datos por parte de la vista.
- Inevitable cambio a la vista y controlador cuando se porta el sistema.

REFERENCIAS

http://dc.exa.unrc.edu.ar/nuevodc/materias/sistemas/2008/Proyecto/1211819913/Anexo06.pdf

ESTRUCTURA



PATRON: LAYERS

CATEGORIAS

DESCRIPCION

El estilo arquitectural en capas se basa en distribución jerárquica de los roles y de las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades funcionales que implementan

VENTAJAS

- ✓ Abstrae ya que los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que usa en cada capa del modelo.
- ✓ Aislamiento ya que se pueden realizar actualizaciones en el interior de las capas sin que esto afecte el resto del sistema.
- ✓ Rendimiento ya que distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallas y el rendimiento.
- ✓ Testeabilidad ya que cada capa tiene una interfaz bien definida sobre que realizar las pruebas y la habilidad de cambiar entre diferentes implementaciones de una capa.
- ✓ Independencia ya que elimina la necesidad considerar hardware y el despliegue así como las dependencias con interfaces externas.

DESVENTAJAS

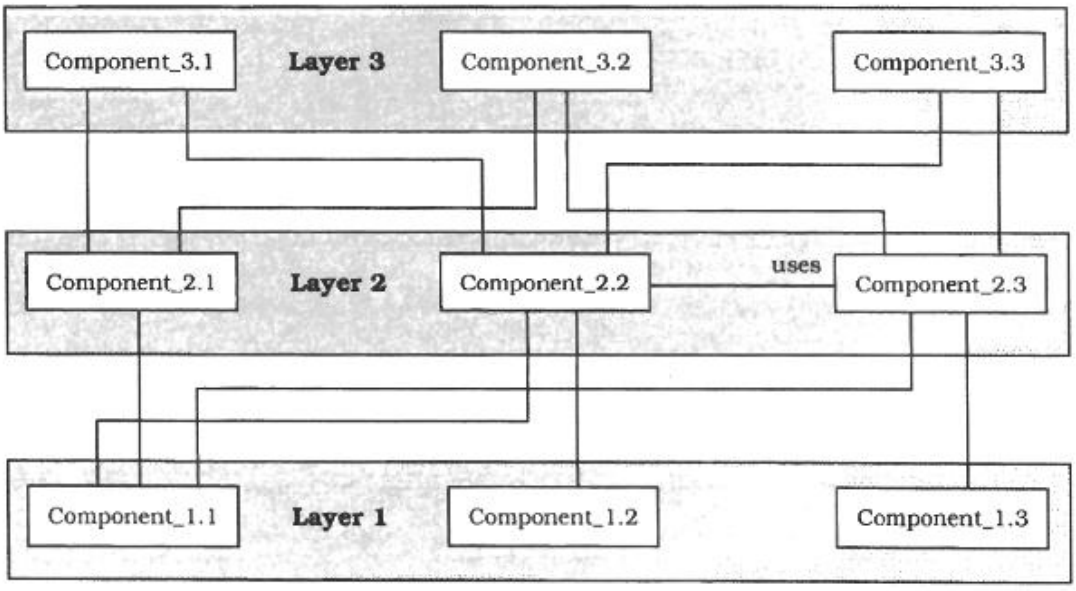
- ✓ Produce cambios de comportamiento en cascada: Suele darse ésta circunstancia y da un problema considerablemente complejo cuando un Layer cambia de comportamiento.
- ✓ Reduce la eficiencia: En ciertos casos la estructura rígida de éste patrón reduce la eficiencia, se da cuando una capa de más alto nivel requiere servicios de la última capa. La solicitud de un servicio que provee la última capa, en forma directa a la capa superior, debe atravesar toda la pila, para que luego, si es necesario la respuesta recorra exactamente el mismo camino.
- ✓ Dificultad para determinar la granulidad.

REFERENCIAS

http://www.vico.org/pages/PatronsDisseny/Pattern%20Layers/index.html

Gamma, E., Helm, R., Johnson, R. and Vlissedes, J. (1995) *Design Patterns:Elements of Reusable Object-Oriented Software*, Reading MA: Addison-Wesley Gardner, K., Rush, A., Crist, M., Konitzer, R. and Teegarden, B. (1998) *Cognitive*

ESTRUCTURA



PATRON KERNEL

CATEGORIAS	Sistemas Adaptables

DESCRIPCION

El patrón arquitectural en cuestión se aplica a sistemas de software que deben poder adaptarse a requerimientos de software cambiantes. Separa un kernel1 funcional mínimo de funcionalidad extendida. Esto es, la parte más básica se posiciona sobre un componente pequeño, que interactúa, controla, y direcciona sobre los componentes que explotan toda la funcionalidad del sistema. La idea es concentrar el sistema en procesos separados que interactúan entre sí. Debido a este mecanismo, veremos más adelante que este patrón tiene desventajas muy importantes, a tal punto que no existe ningún sistema operativo funcional desarrollado con el mismo (Si bien existen algunos como, "Mach", "Amoeba","GNU/Hurd", "Minix", o bien algunos son sólo intenciones teóricas o aquellos llevados a la práctica son complicados de desarrollar, e.g.,"GNU/Hurd" empezado en 1990 y todavía sin una buena implementación). El problema de éste tipo de diseño, es la cantidad de situaciones que emergen debido a la concurrencia entre procesos. Ninguno de los Sistemas nombrados arriba lo manejan fácilmente. Si bien la idea del patrón apunta a ser una de las mejores opciones a la hora de optar por un diseño de un sistema operativo, todavía existen pequeños problemas por solucionar (algunos no tan pequeños).

VENTAJAS

- ✓ Una de las principales ventajas de éste patrón es la portabilidad que se lograría si fuese más directa su implementación. La colaboración entre procesos y los problemas que surgen debido a la concurrencia existente lo hacen muy difícil. Cabe destacar que éste patrón es el ideal para desarrollar un sistema, pero siempre se vuelve al mismo punto, los sistemas implementados con ésta estructura son todavía inestables.
- ✓ Otra ventaja para destacar es la flexibilidad que se obtiene al aplicar el patrón. Esto se debe a que los servidores externos pueden actuar como plugins. Una vez definido el microkernel, sólo basta agregar vistas al sistema. De ésta manera se puede extender el sistema ampliamente. Volviendo siempre sobre el mismo punto, si hablabamos de problemas de concurrencia, al agrandar cada vez más el sistema, se obtienen más situaciones inmanejables, y más complejidad

DESVENTAJAS

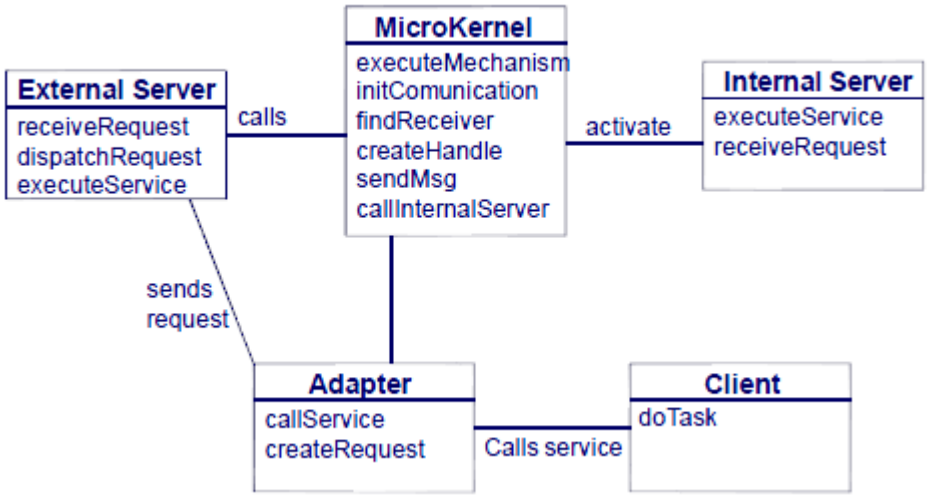
- ✓ se pierde rendimiento al tener tantos procesos separados e integrarlos a traves de una comunicación estable, esto conlleva a una segunda desventaja que es nada menos que la complejidad que se obtiene por intentar coordinar tantos procesos por separado.

REFERENCIAS

Buschmann, F., y otros. 1996. Pattern Oriented Software Architecture: A System of Patterns. Inglaterra : John Wiley & Sons, 1996. 0471958697

http://dc.exa.unrc.edu.ar/nuevodc/materias/ingenieria/Material/1163109537/1163109830/microkernel%202.pdf

ESTRUCTURA



PATRON: BLACKBOARD

CATEGORIAS	Sistemas basados en datos

DESCRIPCION

Es útil para problemas en los cuales ninguna estrategia para obtener soluciones en forma determinística es conocida. Varios subsistemas especializados ensamblan su conocimiento para construir una solución aproximada posible

VENTAJAS

- ✓ Experimentación: en dominios en los que ningún acercamiento al espacio de soluciones es factible, este patrón es apto, ya que experimenta con vario posibles algoritmos en la búsqueda de una solución.
- ✓ Soporte para los cambios y mantenimiento: la fuentes del conocimiento individuales, el componente de control y la estructura de los datos central, se encuentran separadas, sin embargo todos los módulos pueden comunicarse vía blackboard.
- ✓ Reusabilidad de las fuentes del conocimiento: esto es debido a que ellas son independientes unas de otras y se especializan en determinada tarea.
- ✓ Soporte para tolerancia de errores y robustez: en una arquitectura blackBoard todos los resultados son solo hipótesis. Solo aquellos que son apoyados fuertemente or datos y otras hipótesis sobreviven.

DESVENTAJAS

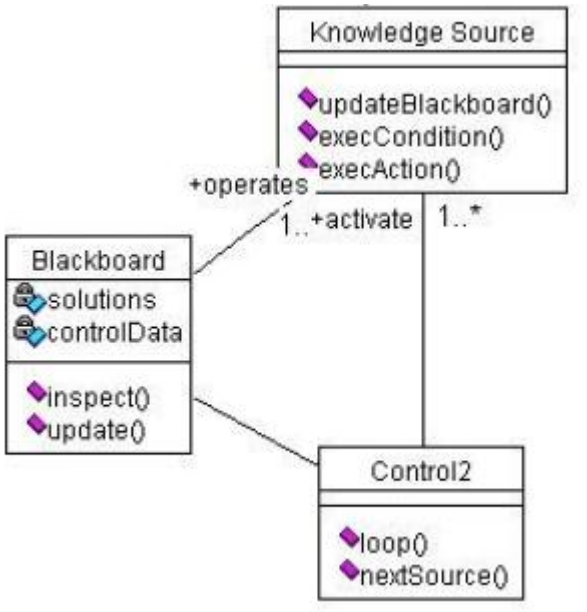
- ✓ Dificultad de comprobación: desde los cómputos de un sistema blackboard no sigue un algoritmo determinístico, sus resultados no son a menudo reproducibles. Además, las hipótesis incorrectas son parte de los procesos de la solución.
- ✓ Ninguna solución buena es garantizada: Normalmente los sistemas blackboard pueden resolver solo un cierto porcentaje de sus tareas correctamente.
- ✓ Dificultad de establecer una buena estrategia de control: la estrategia de control no puede diseñarse de una manera concreta, y requiere un acercamiento experimental.
- ✓ Baja eficacia: se debe a la inexistencia de algoritmo determinístico que solucione el problema.
- ✓ Alto esfuerzo en desarrollo: la mayor de los sistemas Blackboard tardan años para evolucionar, se atribuye esto a los dominios del problema mal estructurados, y el ensayo y error extenso que se programa al definir el vocabulario, la estrategia de control y las fuentes de conocimiento.

REFERENCIAS

http://dc.exa.unrc.edu.ar/nuevodc/materias/ingenieria/Material/1163109537/1163109549

Buschmann, F., y otros. 1996. Pattern Oriented Software Architecture: A System of Patterns. Inglaterra : John Wiley & Sons, 1996. 0471958697

ESTRUCTURA



PATRON PIPES Y FILTERS

CATEGORIAS	Sistemas basados en flujos

DESCRIPCION

En este estilo arquitectónico cada componente tiene un conjunto de entradas y uno de salida. Un componente
lee un flujo de datos en la entrada y produce un flujo de datos diferente en su salida. Esto es logrado
aplicando una transformación local al flujo de entrada mientras éste se lee, de tal forma que el flujo de salida
empieza antes que se consume todo el flujo de entrada. Por lo tanto a los componentes se los denomina
filtros. Los conectores sirven como conductos para transmitir las salidas de un filtro a las entradas de otro,
por lo cual se les denomina tuberías.

VENTAJAS

- ✓ Permite entender la E/S como composición de filtros.
- ✓ Procesamiento paralelo eficiente.
- ✓ Soportan la ejecución concurrente (filtros distribuidos en procesadores concurrentes).
- ✓ Mantenimiento y reuso.
- ✓ - flexibilidad mediante cambios de filtros.
- ✓ - Facilidad de diagnóstico (rendimiento).

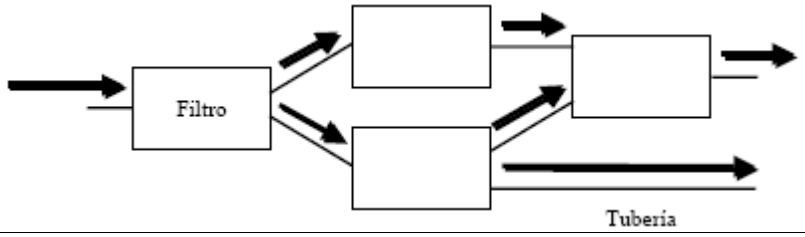
DESVENTAJAS

- ✓ Los datos deben ser transformados para enviar por las tuberías (hay costo de transformación de datos).
- ✓ Las aplicaciones interactivas son complejas (orientados al procesamiento secuencial, no interactivo).
- ✓ el costo del paralelismo puede ser muy grande.

REFERENCIAS

Buschmann, F., y otros. 1996. Pattern Oriented Software Architecture: A System of Patterns. Inglaterra : John Wiley & Sons, 1996. 0471958697

ESTRUCTURA



PATRON: BROKER

CATEGORIAS	Sistemas distribuidos

DESCRIPCION

El bróker es una patrón de arquitectura que se utiliza para estructurar sistema de software distribuidos con componentes desacoplados que interactúan por invocaciones de servidos remotos
Los componentes deben ser capaces de accesar a los servicios proveídos por otros a través de invocaciones de servicios remotos transparentes en ubicación
Se necesita intercambiar, añadir y quitar componentes en tiempo de ejecución.
La arquitectura debe esconder los detalles específicos de implementación del sistema de los usuarios de componentes y servicios.

VENTAJAS

- ✓ Transparencia en ubicación.
- ✓ Cambio y extensibilidad de los componentes
- ✓ Portabilidad del sistema bróker
- ✓ Interoperabilidad entre diferentes sistemas bróker
- ✓ Reusabilidad

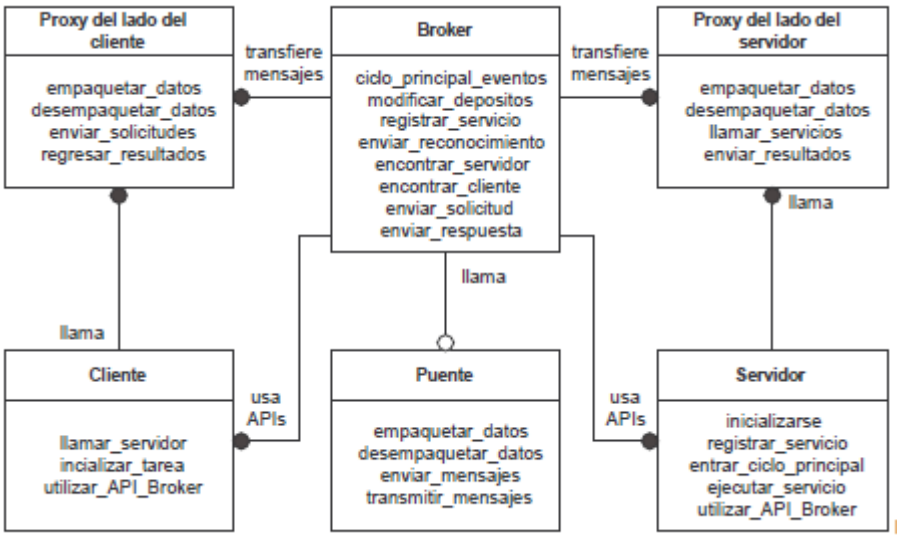
DESVENTAJAS

- ✓ Eficiencia restringida: son más lentas que las aplicaciones cuya distribución de componentes es estática y conocida.
- ✓ Baja tolerancia a fallos.
- ✓ Prueba y debbugin.: es tediosa por su número de componentes es alto.

REFERENCIAS

Buschmann, F., y otros. 1996. Pattern Oriented Software Architecture: A System of Patterns. Inglaterra : John Wiley & Sons, 1996. 0471958697

ESTRUCTURA



Thin Web Client

CATEGORIAS	WEB

DESCRIPCION

Las lógicas de negocios son ejecutadas en el servidor durante el procesamiento de páginas requeridas por el browser del cliente.

Requiere que le cliente tenga un navegador Web estándar capaz de mostrar formularios. Toda la lógica del negocio se ejecuta en el servidor durante el procesamiento de páginas requeridas por el browser del cliente.
Existe poco control por parte del cliente.
Los elementos del servidor se tratan igual que los de los sistema cliente/servidor.
La lógica de la aplicación solo se ejecuta solo se ejecuta si realizan peticiones por parte del cliente y solo en el procesamiento de la petición.
Tiene la limitante de usar interfaces de usuario relativamente simples

VENTAJAS

- Al tener que usar interfaces de usario relativamente simples, no se malgastan esfuerzos en hacer interfaces atractivas cuando con otras normales es suficiente.
- Resultan mucho más funcionales con conexiones de red no muy rápidas.

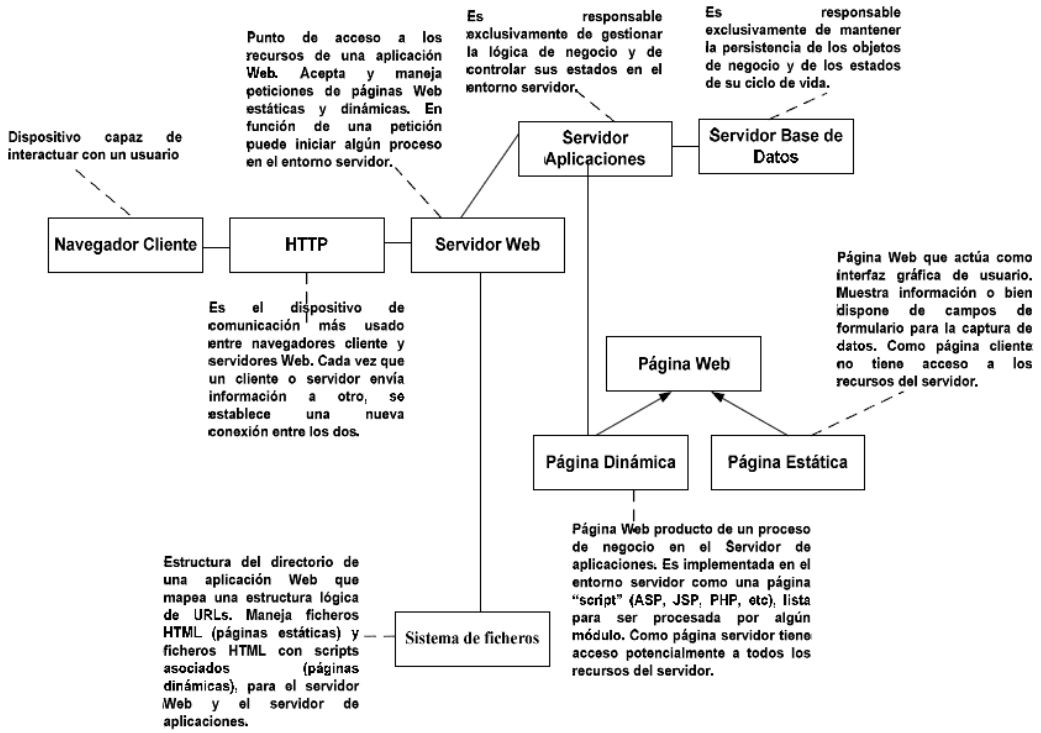
DESVENTAJAS

- Dificultad para manejar interfaces de usuario, por que solo se limita al uso del HTML (texto, algunos botones, campos).
- Este tipo de arquitectura solo conviene para aplicaciones donde el servidor puede responder dentro esperado por el usuario y dentro del tiempo fuera(time out) permitido por los valores de los usuarios(No mayor de unos cuantos segundos)

REFERENCIAS

http://dc.exa.unrc.edu.ar/nuevodc/materias/sistemas/2008/Proyecto/1211819913/Anexo06.pdf
http://dspace.espoch.edu.ec/bitstream/123456789/98/1/18t00375.pdf

ESTRUCTURA



CLIENTE WEB PESADO(THICK WEB CLIENT)

CATEGORIAS	WEB

DESCRIPCION

Una cantidad significativa de la lógica de negocio se ejecuta en la máquina del cliente web ligero con la ubicación de scripts en el entorno cliente y objetos personalizados como ActiveX y applets de java.
Este patrón es el más apropiado en aplicaciones web en el cual una cierta configuración del cliente y una cierta versión de buscador pueden ser asumidas desde la lógica del cliente.

VENTAJAS

- La actividad más simple en este tipo de arquitecturas es comprobar la validez de los campos de un formulario.

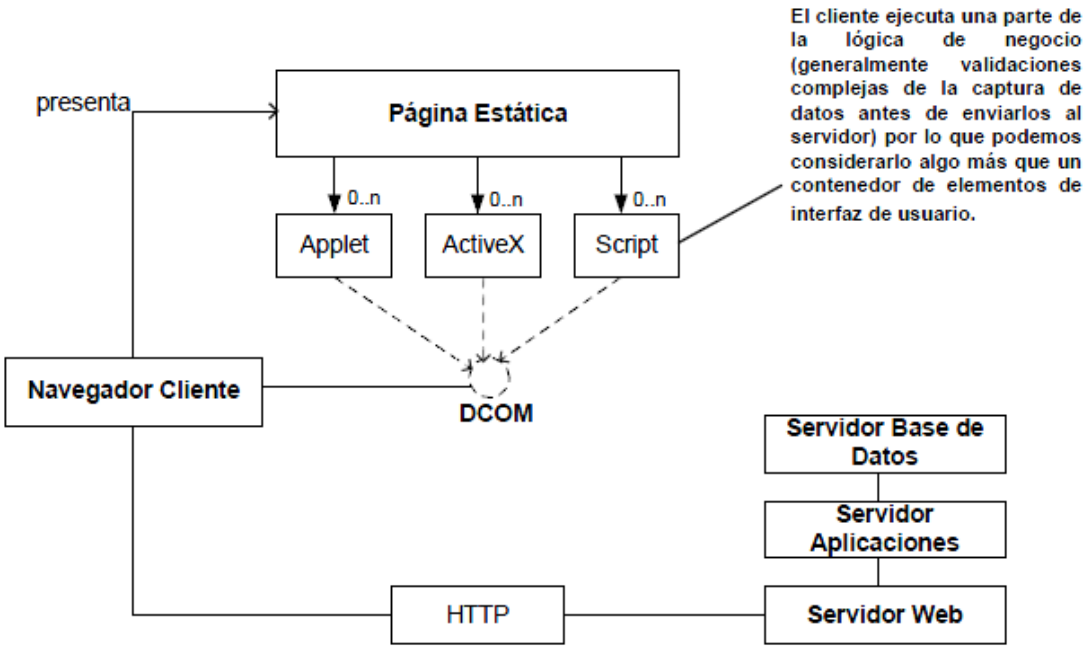
DESVENTAJAS

- No es 100% compatible con los servidores.
- No hay que asumir que el comportamiento en todos los navegadores nunca será el mismo, ma aun en distintos sistemas operativos.
- Incompatibilidad con los navegadores, no todos los navegadores soportan javascripts o vbScripts .
- Implementación del DOM en algunos casos depende del navegador.
- Es necesario hacer pruebas en todos los posibles escenarios para cada cliente que lo soportará y sobre todo los navegadores que soportan la lógica.(Nunca asuma el comportamiento de un navegador)

REFERENCIAS

http://dc.exa.unrc.edu.ar/nuevodc/materias/sistemas/2008/Proyecto/1211819913/Anexo06.pdf
http://dspace.espoch.edu.ec/bitstream/123456789/98/1/18t00375.pdf

ESTRUCTURA



DISTRIBUCIONES WEB (WEB DELIVERY)

CATEGORIAS	WEB

DESCRIPCION

Es denominado de esta forma ya que es utilizado en la web como un mecanismo de distribución de objetos en un sistema cliente/servidor. Desde una perspectiva, este tipo de aplicaciones son distribuidas entre cliente/servidor ya que tiene lugar para anexar un servidor web a un cliente navegador como un elemento crítico de la arquitectura

.

Si tal sistema es una aplicación web con objetos distribuidos o un sistema con objetos distribuidos con elementos web, estos últimos sistemas son los mismos. El hecho que estas dos perspectivas son las mismas en los sistemas es que los sistemas de objetos distribuidos siempre son vistos como sistemas carecidos de un modelo cuidadoso

Usa protocolo HTTP para soportar objetos distribuidos

El navegador funciona como una interfaz para un sistema de objetos distribuidos.

Los elementos más significativos que incluye son : DCOM, IIOP o RMI(JRMP)

VENTAJAS

- ✓ Disponibilidad de conexiones permanentes.
- ✓ Se suele usar junto con otras arquitecturas.
- ✓ La ventaja de usar un navegador es que incorpora una serie de características útiles que no tenemos que re implementar, como la descarga desde el servidor web de los componentes necesarios para ejecutar la aplicación, lo que hace innecesario, por ejemplo, el proceso de instalación del cliente.

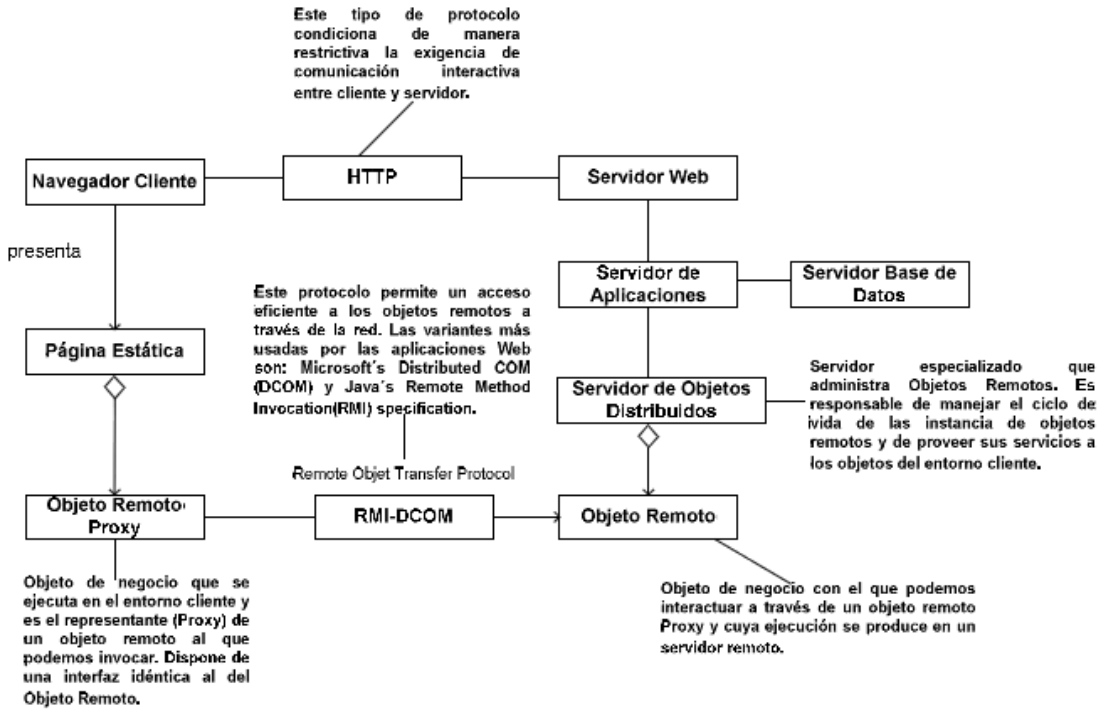
DESVENTAJAS

- ✓ No es una arquitectura muy adecuada para aplicaciones que funcionen en internet.
- ✓ Una desventaja de esta tecnología es que la portabilidad disminuye notablemente.
- ✓ Para su implementación es necesario una red bastante fiable.
- ✓ Gastan mucho en conexiones HTTP entre el cliente y el servidor y esporádicamente se pierde el servidor, lo cual no es un problema para los patrones anteriores.

REFERENCIAS

http://dc.exa.unrc.edu.ar/nuevodc/materias/sistemas/2008/Proyecto/1211819913/Anexo06.pdf

ESTRUCTURA



Patrones de Análisis

Party (Grupo)

CATEGORIAS	Responsabilidad

DESCRIPCION

Se aplica cuando se tiene varios objetos que tienen las mismas actividades o atributos.
En el diseño será aplicado como una herencia.

VENTAJAS

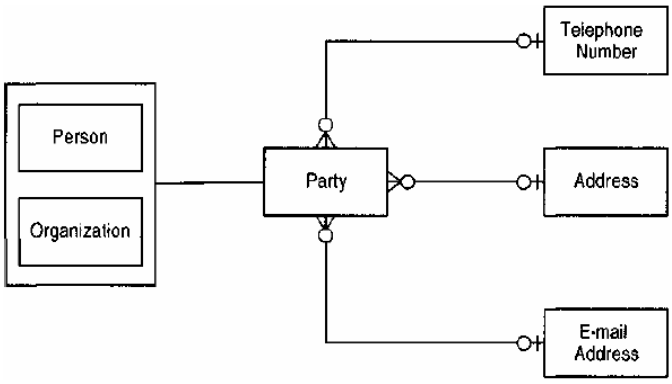
Permite definir súper tipos.
Da mayor abstracción en función de la asociación.

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Organization Hierarchies (Organización de Jerarquía)

CATEGORIAS	Responsabilidad

DESCRIPCION

Cuando existe una estructura jerárquica en la cual un objeto depende de otro, se aplica este patrón que tiene como objetivo agrupar todos los componentes dependientes que son dirigidos por uno nuevo que organiza y jerarquiza los objetos agrupados.

VENTAJAS

Mayor flexibilidad en la estructuración de responsabilidad para la organización.

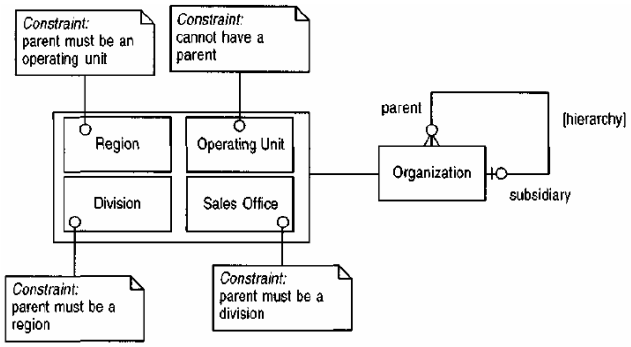
DESVENTAJAS

Soporte una sola organización, pude existir funcionamientos diferentes para diferentes grupos del negocio.

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Organization Structure (Organización de Estructura)

CATEGORIAS	Responsabilidad

DESCRIPCION

Crea una estructura que permite instancias de los elementos creados.

Cada relación entre las organizaciones está definida por un tipo de estructura de la organización. Es mejor que explícitamente la asociación si hay muchas relaciones.

VENTAJAS

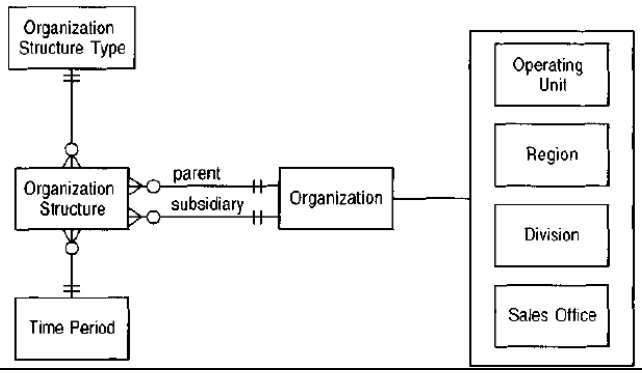
Facilidad para exponer la posibilidad de crear varias y diferentes instancia

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Accountability (Responsabilidad)

CATEGORIAS	Responsabilidad

DESCRIPCION

Permite definir una estructura de responsabilidad para que sea asociada a los componentes necesarios.

Usa un Party para poder asignar un conjunto de responsabilidades a un conjunto de objetos..

VENTAJAS

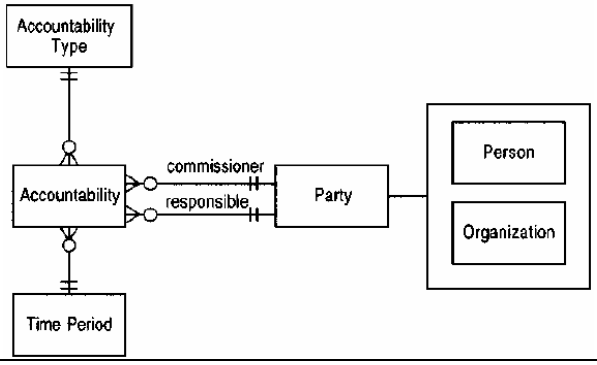
DESVENTAJAS

Asigna responsabilidad a todo el grupo, sin poder especificar y es un solo elemento del grupo.

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Accountability Knowledge Level

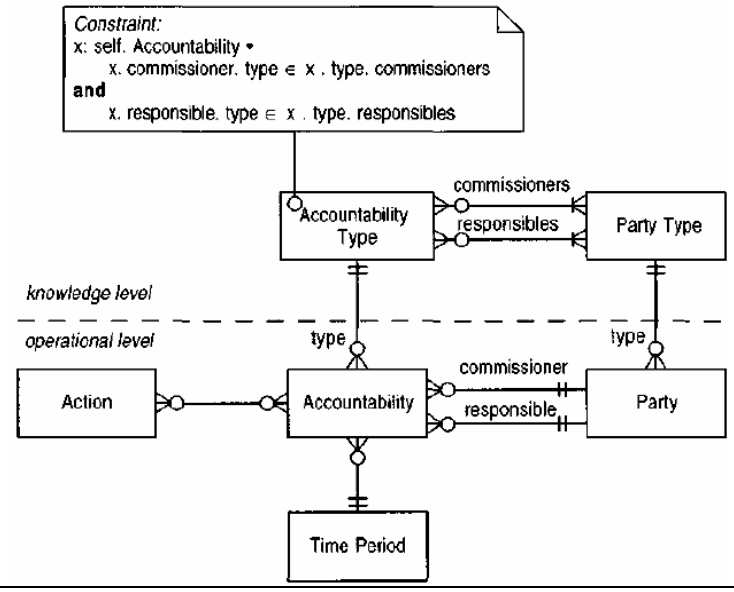
CATEGORIAS	Responsabilidad

DESCRIPCION	VENTAJAS
Se aplica para delegar tipos de responsabilidad a un Party. Genera una asociación a nivel de objetos entre Party y Accountability	En este ya se pude determinar cuál es el conjunto y cuál es la responsabilidad asociada
	DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Party Type Generalizations (Generación de tipo Grupo)

CATEGORIAS	Responsabilidad

DESCRIPCION

Permite la generalización de tipos de grupo que heredan de un subtipo

Ayuda a especificar que un empleado puede ser también cliente, siendo el tipo persona y los subtipos empleado y cliente.

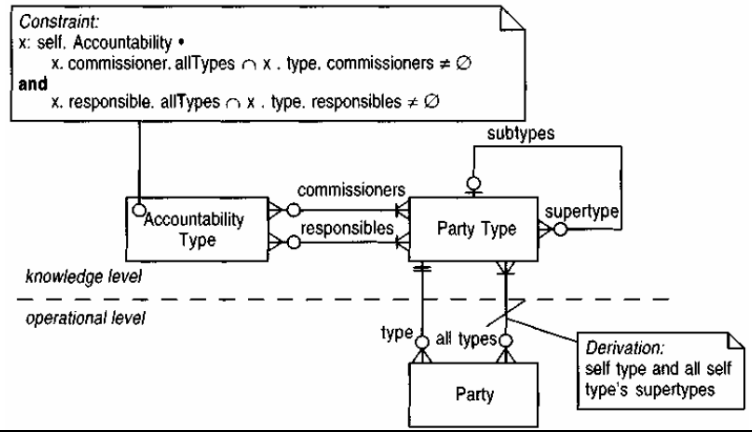
VENTAJAS

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Hierarchic Accountability (Jerarquía de Responsabilidad)

CATEGORIAS	Responsabilidad

DESCRIPCION

Agrega restricciones a los elementos de responsabilidad. Estos describen las restricciones de los contratos entre los componentes

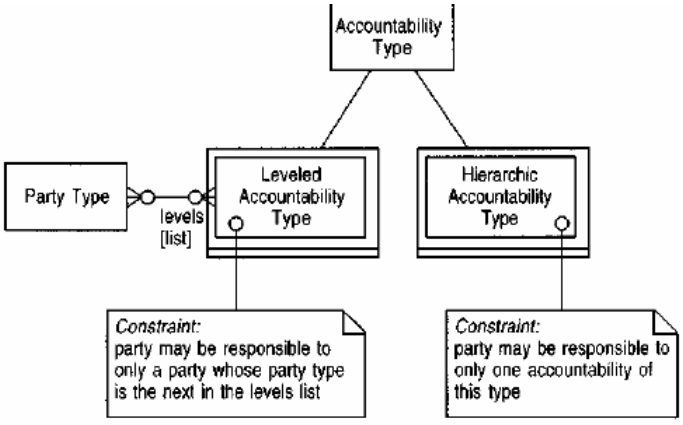
VENTAJAS

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Quantity (Cantidad)

CATEGORIAS	Observación

DESCRIPCION

El objetivo es medir los atributos usando Cantidades.
Este se puede usar cuando el atributo lo requiera.

Asocia los atributos de una clase a un cantidad que permita conocer el formato del atributo.

VENTAJAS

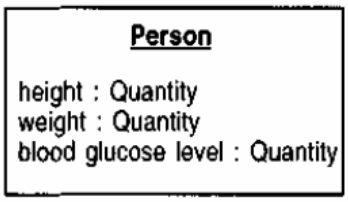
Este ayuda a establecer los datos para el diseño dando un referente de tipo, tamaño y valores para los atributos de las clase

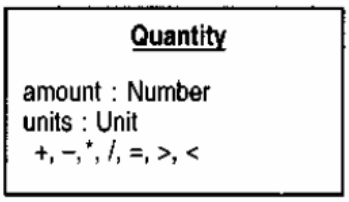
DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA





Conversion Ratio (Razón de Conversión)

CATEGORIAS	Observación

DESCRIPCION

Permite exponer la necesidad de realizar una conversión, es como tomar un valor atómico que al pasar por el conversión ratio se transforma en un valor compuesto.
Ejemplo convertir un número a un dato monetario

VENTAJAS

Ayuda a determinar la extensibilidad de los datos básicos.

Define un glosario de datos

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA

Compound Units (Unidades Compuestas)

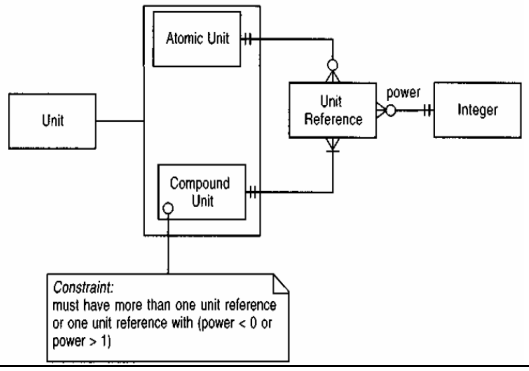
CATEGORIAS	Observación

DESCRIPCION	VENTAJAS
Este toma las unidades o las razones de conversión y lo transforma en datos compuestos, por ejemplo venta por metro cuadrado.	
	DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Measurement (Medición)

CATEGORIAS	Observación

DESCRIPCION

Este patrón realiza observaciones de las categorías y los fenómenos

Pero muchas observaciones consisten simplemente en una declaración de ausencia o la presencia en lugar de un rango de valores.

Ejemplo Podemos modelar un bajo nivel de aceite en un automóvil como una observación de la categoría del coche. El tipo de fenómeno es el nivel de aceite con posibles fenómenos de más de lleno, está bien, y baja. La observación de los vínculos del coche con el fenómeno de baja.

VENTAJAS

Permite validar las posibles asociaciones de las personas con su diferentes categorías y su medición

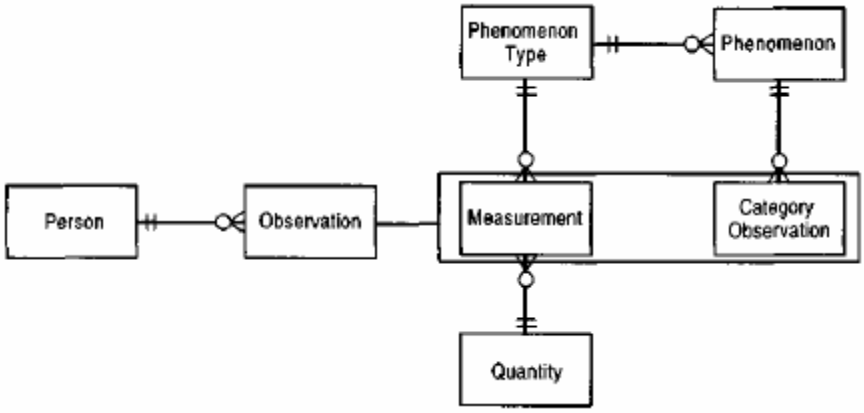
DESVENTAJAS

Si son mucha las observaciones puede tener un difícil nivel de abstracción

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



Observation

CATEGORIAS	Observación

DESCRIPCION

Este patrón se encarga de asignara propiedades de medición a los tributos que estén asociados a una cantidad

Brinda una herramienta para poder asignar en un atributo de cantidad una medición, Ejemplo John Smith es de 6 pies de altura, que pueden ser representados por una medida cuya persona es John Smith, tipo de fenómeno es la altura, y la cantidad es de 6 pies.

Mide un tipo fenómeno usando la cantidad para una persona.

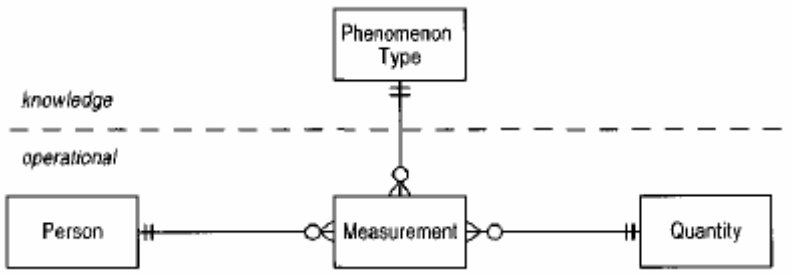
VENTAJAS

DESVENTAJAS

REFERENCIAS

Libro Addison Wesley – Analysis

ESTRUCTURA



LowCoupling/High Cohesion (Bajo Acoplamiento/Alta Cohesion)

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION

EL patrón sugiere crear más clases que sean especializadas y asignar toda la responsabilidad a una sola.

VENTAJAS

Reusabilidad de clases
Facilidad de mantenimiento
Fundamento para aplicación de otros patrones

DESVENTAJAS

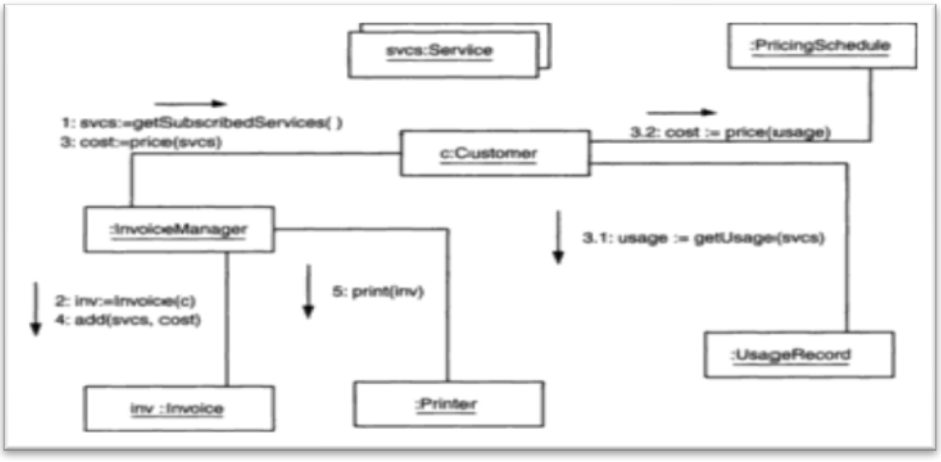
Se deben crear mas objetos para realizar la misma tarea, se tiene que entrar a ver el rendimiento

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Inicialmente el Customer tenía mayor acoplamiento por queInvoice y Printer eran instanciados por él, se baja el acoplamientos creado mayor cohesión a través deInvoiceManager.



Expert (Experto)

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION

Se le asigna la responsabilidad a la clase que tenga la información suficiente para poder tener la responsabilidad.

VENTAJAS

Aplicar bien este patrón ayuda hacer más extensibles los diseños, porque se lo logra una buena encapsulación.

DESVENTAJAS

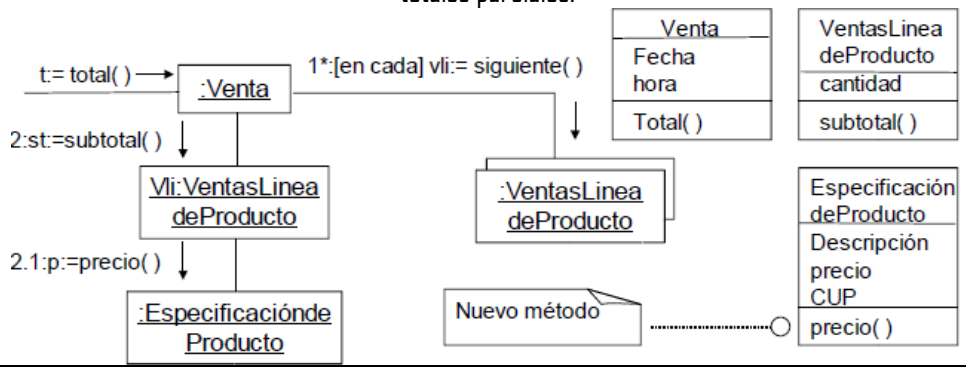
Puede desarrollar un alto acoplamiento, cuando se crea algo así como un monopolio es decir que se le delega mucha responsabilidad al experto

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Se desea conocer el valor total de una venta por cada line se produce un valor pero el experto es venta que es que conoce los totales parciales.



Creator (Creador)

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION	**VENTAJAS**
Determina cual es la clase responsable de crear instancias de otra.	Definiendo bien la clase creador se vitan tener aseo por referencia, pues se usa la instancia para conocer el objeto.
	DESVENTAJAS

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Class B should be responsible for creating instances of Class A if any of the following are true:

- Class B and Class A are the same class and their instances compose, aggregate, contain, or directly use other instances of the same class.
- Instances of Class B compose or aggregate instances of Class A.
- Instances of Class B contain instances of Class A.
- Instances of Class B record instances of Class A.
- Instances of Class B directly use instances of Class A.
- Instances of Class B have the data that is passed to constructors of Class A. Thus, the Expert pattern suggests making Class B responsible for creating instances of Class A.

When more than one of these relationships apply, choosing the one closer to the top of the list will usually produce the better result.

Polymorphism (Polimorfismo)

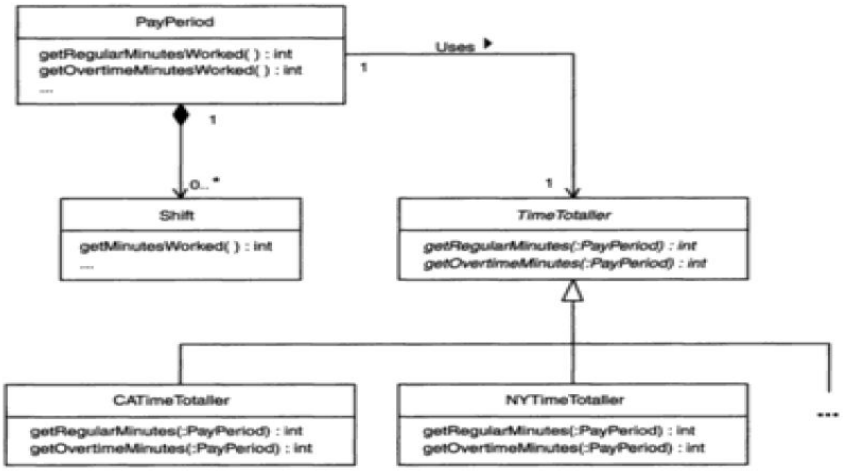
CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION	**VENTAJAS**
Cuando se cuenta con diferentes implementaciones de una misma clase se cuenta con un objeto base para crear las diferentes implementaciones.	Alto nivel de abstracción
	DESVENTAJAS

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA



Pure Fabrication (Fabrica Pura)

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION

Delegar responsabilidad en otra clase para bajar el acoplamiento de una clase que tenga mucha responsabilidad.
En Comparación con GOF el Adapter, Command y Strategy usan este patrón.

VENTAJAS

Baja acoplamiento

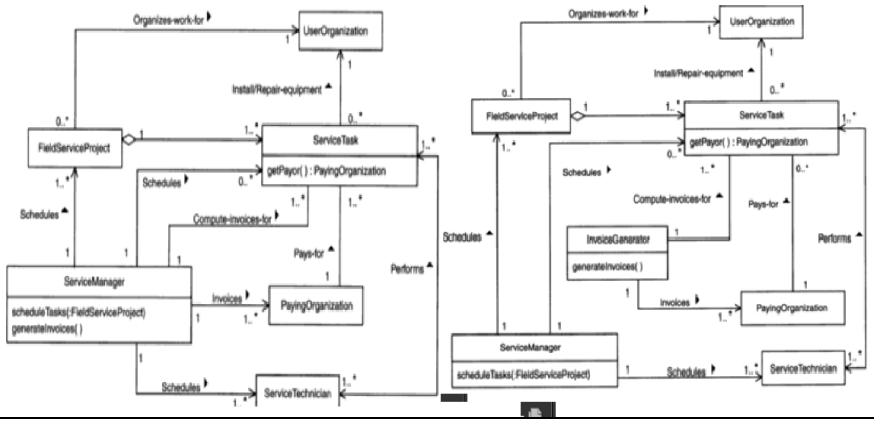
DESVENTAJAS

Puede llegar a ser desordenada la delegación de responsabilidad, pues el objetivo es bajar acoplamiento.

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA



Law of Demete

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION

Hablar con las clases más cercanas,
Este patrón hace que las clase estén menos acopladas haciendo que a través de métodos que permiten hacer mas directa la comunicación entre clases..

VENTAJAS

Baja acoplamiento

DESVENTAJAS

La aplicación con celo excesivo conduce a un exceso de métodos en las clases intermedias.

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Ley de Demeter - (Lieberherr, Holland y Riel 1988), también llamada regla de "No hables con extraños". Para toda clase C y para todos los métodos M asociados a C, todos los objetos con los cuales M interactúa deben ser :

- Instancias de las clases a los que pertenecen los argumentos de M
- Instancias de C
- Objetos creados directamente por M, o por los métodos a los que M invoca.
- Objetos globales.

Otra forma de expresar la Ley de Demeter es mediante la analogía de los juguetes:

- Puedes jugar contigo mismo
- O con los juguetes que te han dado
- O con los juguetes que construyas tú.

Controller

CATEGORIAS	GRASP (General Responsability Assignment Software Patterns)

DESCRIPCION

Se relaciona con el Facade de GoF

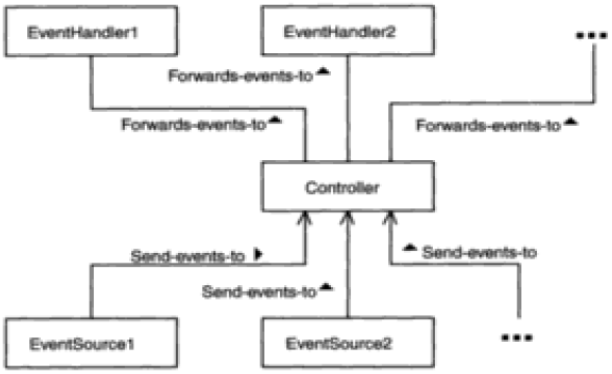
VENTAJAS

DESVENTAJAS

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA



Window per Task

CATEGORIAS	GUI

DESCRIPCION

Organiza la interfaz de usuario dentro de ventanas dedicadas a diferentes tareas.

Se tiene una ventana para cada tarea del usuario. Se debe tener disponibilidad de toda la información necesaria para que el usuario realice la tarea.

VENTAJAS

Facilidad de uso para el usuario por que presenta todo los componentes necesarios en una solo interfaz.

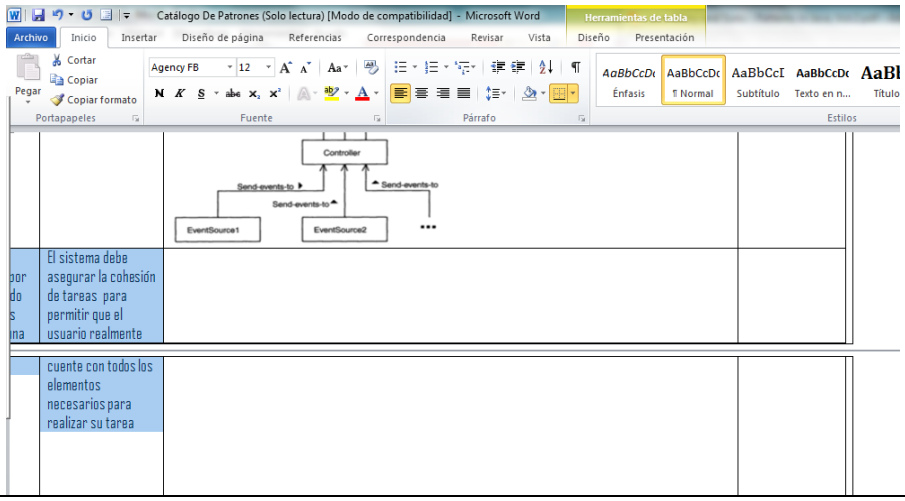
DESVENTAJAS

El sistema debe asegurar la cohesión de tareas para permitir que el usuario realmente cuente con todos los elementos necesarios para realizar su tarea

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA



Interaction Style

CATEGORIAS	GUI

DESCRIPCION

Ayuda para selecciona la forma principal como el usuario interactúa con una ventana.

Este estilo es de los mas usados, se enfoca en cargar datos de la base dato y permite al usuario crear una interfaz de interacción entre usuario y base de datos.

Selección de interacción permiten al usuario seleccionar un valor de datos o la siguiente acción a realizar de la lista

Formulario de interacción permiten al usuario proporcionar información o especifíquese qué hacer rellenando los campos de un formulario

Interacciones de **conversación de texto** permiten a un usuario para entrar comandos complejos como texto

Un estilo de **manipulación directa** de la interacción proporciona una representación visual de la tarea.

VENTAJAS

DESVENTAJAS

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Disabled Irrelevant Thing

CATEGORIAS	GUI

DESCRIPCION	VENTAJAS
Orienta en como ocultar o desactivar elementos GUI que no son relevantes para el contexto actual.	Hace menos pesada la interfaz para el usuario.
	DESVENTAJAS
	Carga componentes que de pronto no se vayan a usar nunca

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

Supplementaty Window Dialog (Ventana Suplementaria Dialogo)

CATEGORIAS	GUI

DESCRIPCION

Ayuda a decidir si una ventana puede ser un dialogo.

Complementa las interacciones del usuario con la ventana principal. Se aplica cuando se debe hacer una función indirecta a la ventana actual.

.

VENTAJAS

Extiende la funcionalidad de la ventana sin tener que cambiar de formulario

Genera baja cohesión por que delega actividades complementarias en otros componentes,

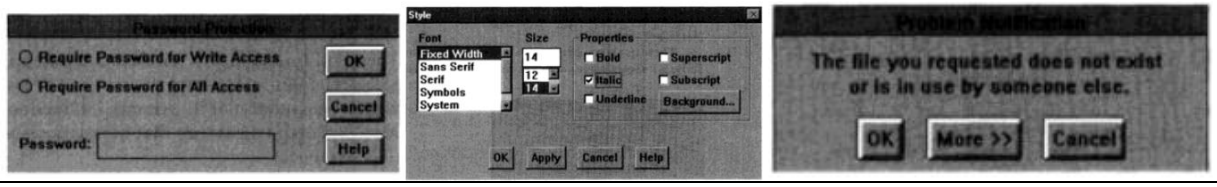
DESVENTAJAS

El exceso de mensajes, hacer la ejecución para el usuario, por que tiene muchos sub-formulario para gestiona

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA



Step by Step Instruction (Instrucciones Paso a Paso)

CATEGORIAS	GUI

DESCRIPCION

Muestra cómo llevar un usuario a través de diferentes pasos de una tarea, cuando la interfaz gráfica le dice al usuario qué hacer a continuación, en lugar de decirle el usuario a la interfaz gráfica qué hacer a continuación..

.

VENTAJAS

Asegura la información necesaria en pasos siguientes.

Facilidad de uso en usuarios sin experiencia.

DESVENTAJAS

Si en un solo paso existe demasiada información puede ser intimidante debe repartirse en varios pasos.

Debe existir mayor validación, para evitar el salto de pasos.
El usuario puede perder la paciencia si hay muchos pasos.

REFERENCIAS

John Wiley and Sons - Patterns in Java, Vol.2

ESTRUCTURA

...