**# [Decisión 001]: Estilo Arquitectónico**

\* Estado [aceptado]

\* Decisores: [Alberto Pacho-Sergio Martín]

\* Fecha: [2020-11- 18]

**## Contexto y declaración del problema**

El cliente necesita cambiar la arquitectura de su página web. Además, de solicitar diversos componentes de la aplicación, como serían el canal de mensajería, componentes de presentación, lógica de dominio y lógica de base de datos.

**## Impulsores de decisión**

RF1-RF3-RF6-RF5

**## Opciones consideradas**

\* Estilo por Capas.

\* Estilo Cliente-Servidor.

\* Estilo Rest.

**## Resultado de la decisión**

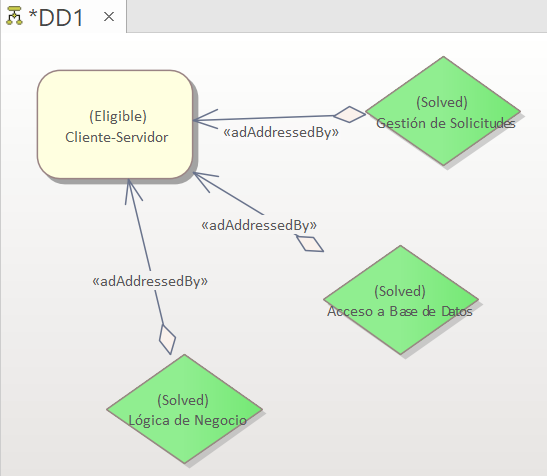
Opción elegida: Cliente-Servidor, porque tras analizar fríamente el problema propuesto, hemos llegado a la conclusión del que el cliente necesita basarse en una arquitectura Cliente-Servidor ya que el estilo se puede dividir por diferentes capas donde se pueden aplicar las necesidades. En este caso la lógica de negocio, la lógica de base de datos y el canal de mensajería.

**### Consecuencias positivas**

\* Facilidad de integración en páginas web.

\* Permite acceder a diferentes bases de datos de una forma dinámica.

**### Consecuencias negativas**



**# [Decisión 002]: Interfaz única**

\* Estado [aceptado]

\* Decisores: [Alberto Pacho-Sergio Martín]

\* Fecha: [2020-11- 18]

**## Contexto y declaración del problema**

Se necesita una interfaz única para acceder a otra interfaz o a otro grupo de interfaces. La interfaz cambia en los diferentes dispositivos que se utilicen y en la forma de comunicarte por mensajería.

**## Impulsores de decisión**

RF3.2 RF2

**## Opciones consideradas**

\* Patrón Facade.

**## Resultado de la decisión**

Opción elegida: “FACADE”, ya que provee de una interfaz única simple para un sistema complejo.

**### Consecuencias positivas**

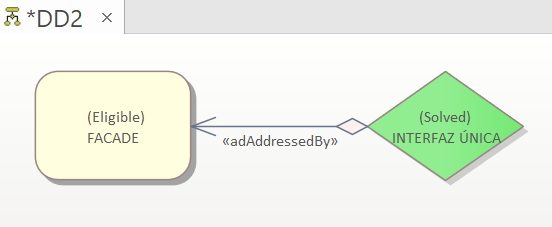
\* Independencia, portabilidad y reutilización.

\* Reducción de dependencias entre subsistemas y los clientes.

\* A la hora de modificar las clases de los subsistemas basta con realizar cambios en la interfaz externa para que los clientes puedan quedar aislados.

**### Consecuencias negativas**

\* Si el acceso de clientes es masivo, podían acabar utilizando solo una parte de la interfaz externa.



**# [Decisión 003]: Una Única Instancia**

\* Estado [aceptado]

\* Decisores: [Alberto Pacho-Sergio Martín]

\* Fecha: [2020-11- 18]

**## Contexto y declaración del problema**

Se tiene la necesidad de crear una única cesta, dado que solo puede existir una compartida para todos los productos, para que el usuario pueda ir añadiéndolos. Además, se requiere que solo pueda haber un único usuario en la sesión iniciada.

**## Impulsores de decisión**

RF-5.4 RF-5.3

**## Opciones consideradas**

\* Patrón Builder.

\* Patrón Sigleton.

**## Resultado de la decisión**

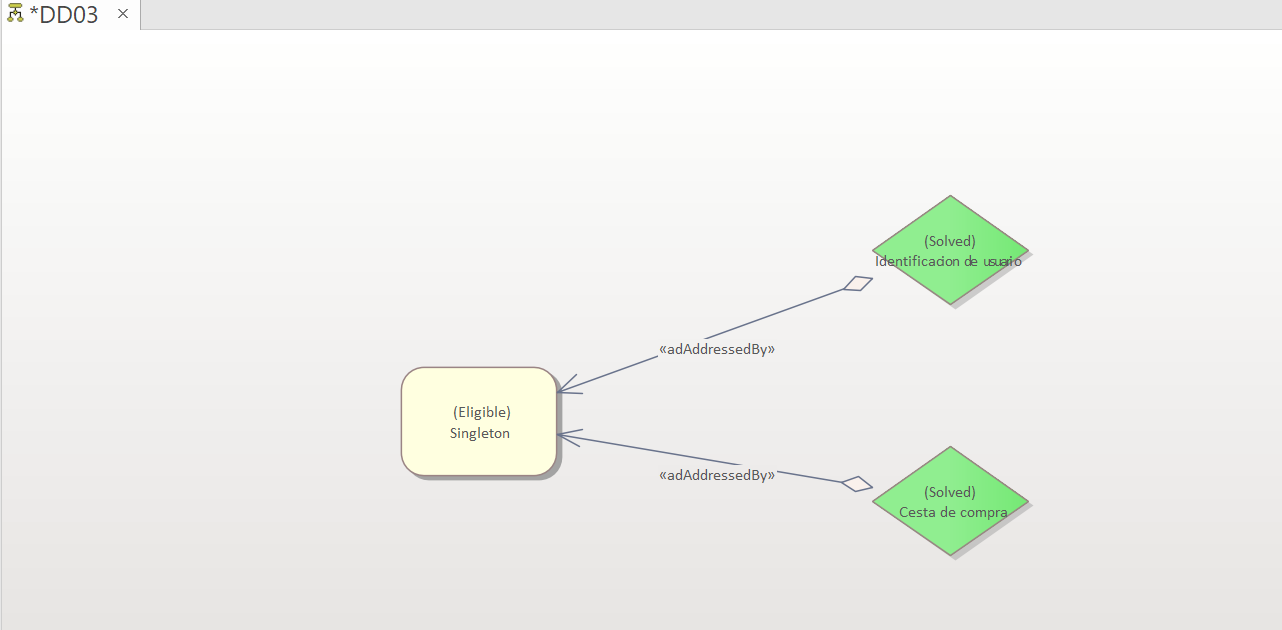
Opción elegida: “SIGLETON”, este patrón garantiza la creación de una única instancia para una clase.

**### Consecuencias positivas**

**Al utilizar este patrón, nos aseguramos de que no haya instancias duplicadas.**

**### Consecuencias negativas**

Dificultad para realizar testing en el programa.



**# [Decisión 004]:**

\* Estado [aceptado]

\* Decisores: [Alberto Pacho-Sergio Martín]

\* Fecha: [2020-25- 18]

**## Contexto y declaración del problema**

Se requiere cierto control de uso y comunicación con la base de datos.

**## Impulsores de decisión**

**RF-3.4**

**## Opciones consideradas**

\* Proxy

**## Resultado de la decisión**

Opción elegida: El patrón “PROXY”. Este patrón proporciona un intermediario de un objeto para controlar su uso. Y de esta forma una clase funciona como una interfaz de otro elemento.

**### Consecuencias positivas**

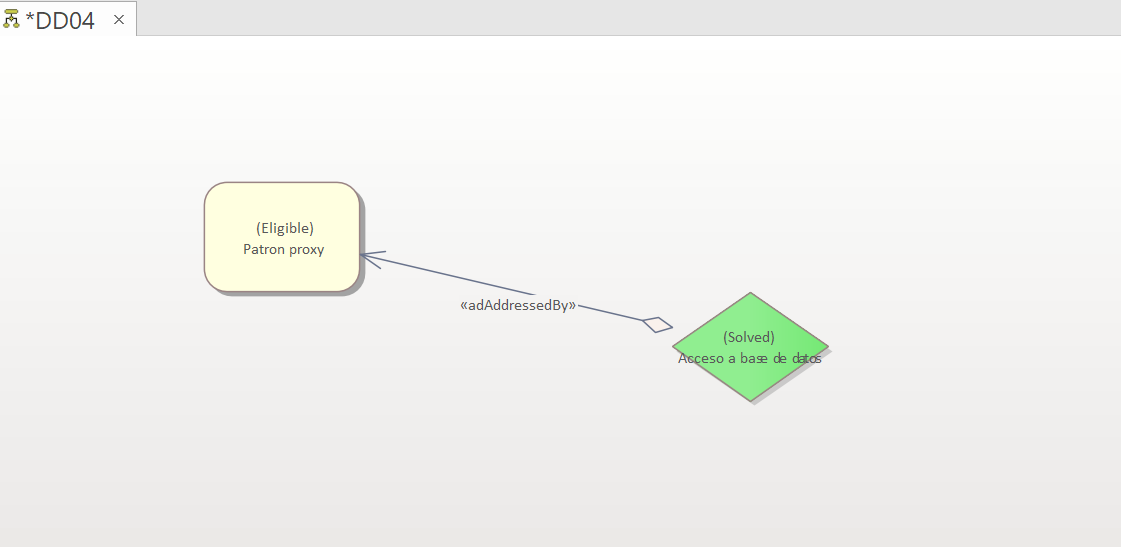
Puedes controlar el objeto de servicio sin que los clientes lo sepan.

Puedes gestionar el ciclo de vida del objeto de servicio cuando a los clientes no les importa.

**### Consecuencias negativas**

El código puede complicarse ya que debes introducir gran cantidad de clases nuevas.

La respuesta del servicio puede retrasarse.



**# [Decisión 005]:**

\* Estado [aceptado]

\* Decisores: [Alberto Pacho-Sergio Martín]

\* Fecha: [2020-25- 18]

**## Contexto y declaración del problema**

Se necesita saber si hay stock disponible de un producto a la hora de realizar un pedido.

**## Impulsores de decisión**

**RF-5.2**

**## Opciones consideradas**

\* Parallel Split

**## Resultado de la decisión**

Opción elegida: patrón “PARALLEL SPLIT”. Este patrón se divide en varios threads que pueden ejecutarse en paralelo permitiendo la ejecución simultanea de varias actividades.

**### Consecuencias positivas**

Permite realizar comprobaciones en tiempo real para asegurarse de que haya stock de un producto a la hora de realizar un pedido.

**### Consecuencias negativas**

Limita el número de hilos del procesador, evitando que otras tareas puedan utilizarlos.

