5. Diseño

En el capítulo anterior se ha realizado un análisis del problema mediante la especificación. De la ejecución de la etapa de especificación se ha generado un output ya que hemos conseguido identificar todas las necesidades que debe satisfacer el editor visual de formularios vía web de manera formal.

A pesar de todo, el output o el resultado de la ejecución de la etapa de la especificación, no es suficiente para poder comenzar a ejecutar la etapa de implementación del sistema. Esta condición de insuficiencia es debida a que el output de la etapa de especificación está orientada hacia la descripción de qué debe hacer el sistema, independientemente de las tecnologías a utilizar, pero no está enfocada a resolver la cuestión de cómo debe hacerlo el sistema.

Por lo tanto una vez identificadas todas las necesidades que debe satisfacer el editor visual de formularios, hay que decidir como se van a poder satisfacer todas éstas necesidades identificadas.

Como hemos detectado es necesario que se ejecute una etapa, entre la etapa de especificación, y la etapa de Implementación. Ésta recibirá como input el resultado procedente de la ejecución de la etapa de especificación, y producirá un output, que permitirá comenzar a ejecutar la etapa de Implementación del editor visual de formularios. Ésta etapa que hemos detectado como necesaria, es la etapa de Diseño.

La etapa de Diseño consiste precisamente en la toma de diversas decisiones, que permitan desarrollar un sistema que sea capaz de resolver satisfactoriamente todas las necesidades identificadas en la etapa de especificación. Si el objetivo de la etapa de Especificación es conseguir detectar que ha de hacer el sistema a desarrollar, entonces el objetivo de la etapa de diseño es determinar que directrices han seguirse para poder desarrollar el sistema, es decir responde a la pregunta, ¿como se tiene que hacer?

En la etapa de Diseño se tienen en cuenta aspectos como la tecnología de la que disponemos. Ésta etapa comenzará con un estudio comparativo entre las posibles arquitecturas.

El objetivo principal de éste capítulo es exponer y justificar las decisiones que se han tomado a la hora de desarrollar el sistema que satisface todas las necesidades detectadas.

Concluyendo, se puede afirmar que la etapa de Diseño se trata de un proceso crucial para poder finalizar con éxito cualquier proyecto.

Gracias al output generado tras la ejecución de la etapa de Diseño, se podrá comenzar a ejecutar la etapa de implementación, donde se empezará a materializar los esfuerzos en forma de sistema de información.

5.1. Diseño arquitectónico

El primer paso en la etapa de Diseño es decidir cual es la arquitectura software más adecuada para nuestra aplicación, es decir la estructura fundamental del sistema.

La estructura del sistema determinará que características tendrá el software que se implementará. Ejemplos de características de software son la eficiencia, la reutilización de código y la fiabilidad del software.

Para poder definir la arquitectura software existen varios patrones arquitectónicos, que ofrecen soluciones generales para lograr que el sistema consiga satisfacer determinadas características software.

La elección de la arquitectura software adecuada para el sistema, suele estar condicionada por la toma de decisiones respecto a los siguientes factores:

- La organización del sistema software.
- La selección de los elementos estructurales y sus interfaces.
- Comportamiento de los elementos estructurales.
- La posible composición de los elementos estructurales en subsistemas más grandes.
- El estilo que rige ésta organización el sistema software.

Antes de empezar a definir la arquitectura software, es necesario identificar las propiedades que deberá cumplir el sistema cuando esté construido.

La identificación de estas propiedades condicionará la elección de los patrones arquitectónicos adecuados para el sistema software.

Ejemplos de propiedades que pueden tener una arquitectura software, con las siguientes:

- ***Reutilizable:*** Es la capacidad que tendrá la arquitectura de reutilizar software ya existente y de producir software, que pueda ser reutilizado.
- ***Probable:*** Es la facilidad que ofrecerá la arquitectura, para realizar pruebas del sistema.
- ***Eficiencia:*** Es la capacidad que tendrá la arquitectura, de realizar una buena utilización de los recursos hardware, de ofrecer un alto rendimiento y de ofrecer unos tiempos de respuesta aceptables.
- ***Fiable:*** Es la capacidad que tendrá la arquitectura de ofrecer una fiabilidad respecto a los fallos del sistema, a pérdidas de conexión y a la robustez frente a la utilización incorrecta del sistema y respecto a tratamientos de errores inesperados.

- ***Interoperable:*** Es la capacidad que tendrá la arquitectura de poder realizar intercambios de funcionalidades y de datos con otras entidades software.
- ***Integridad Conceptual:*** Es la capacidad que tendrá la arquitectura de ofrecer simetría y de poder predecir su comportamiento.
- ***Cambiable:*** La capacidad de cambiabilidad que tendrá la arquitectura de un sistema podemos evaluarla según los siguientes criterios :
    - **Reorganizable**: Es la capacidad de poder reorganizar los diferentes componentes existentes.
    - **Extensible**: Es la capacidad de poder añadir nuevas funcionalidades o mejoras de componentes ya existentes.
    - **Manutención**: Es la capacidad de poder ser mantenido, que abarca la detección y la reparación de errores.
    - **Portabilidad**: Es la capacidad de poder realizar cambios de plataforma hardware, sistemas operativos, lenguajes de programación y sistema de gestores de base de datos.

5.1.1. PATRONES ARQUITECTÓNICOS

Existen varios tipos de patrones arquitectónicos, pero casi todos son clasificables entre los siguientes tipos de patrón:

- ***Modelo Vista-Controlador***

Este tipo patrones se utiliza en sistemas software que interactúan con interfaces de usuario que son muy flexibles, donde la información proporcionada por el sistema y su comportamiento tienen que reflejar inmediatamente las manipulaciones de información producidas.

El Modelo Vista-Controlador también se suele ser útil cuando se desea mostrar la misma información en diversas pantallas o ventanas.

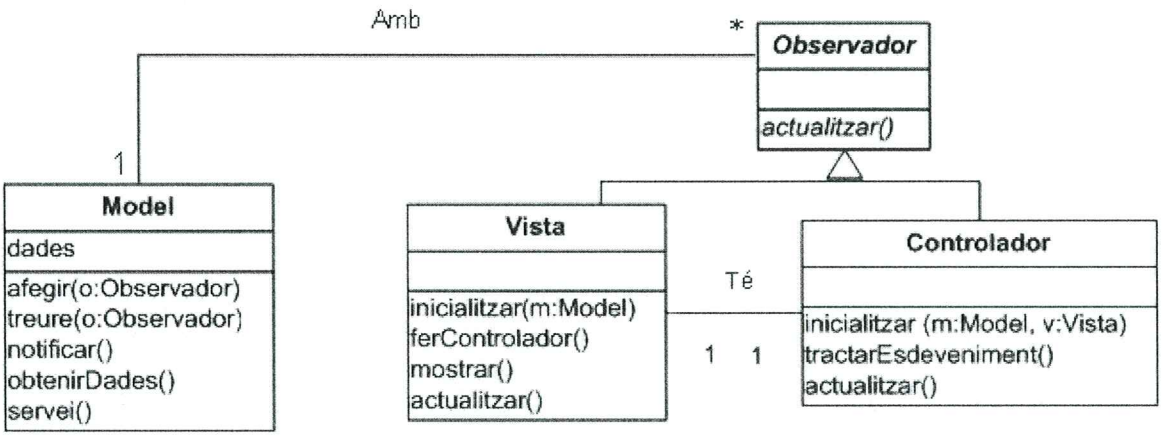
Cómo su nombre indica éste modelo tiene dos componentes:

- El componente Vista, tiene la responsabilidad de gestionar como se muestra la información al usuario.
- El componente Controlador, tiene la responsabilidad de gestionar la interacción con el usuario.

Debido que en los sistemas en los que se aplica el Modelo Vista-Controlador suelen ser críticos, la aplicación de éste patrón resulta poco eficiente respecto al número de recursos necesitado.

También hay que destacar que el uso de éste modelo, incrementa la cambiabilidad y la portabilidad del sistema.

Un ejemplo de la representación gráfica del modelo vista controlador es el siguiente dibujo:



- ***Modelo Llamada y retorno***

Este tipo patrones es el más utilizado todos. Normalmente s utiliza en grandes sistemas de software.

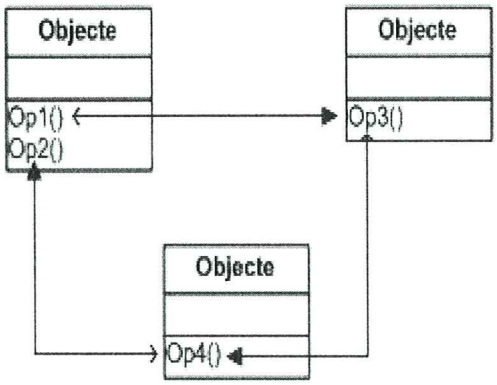
Se pueden distinguir 3 estilos diferentes de este tipo de patrones:

- ***Orientado a objetos***:

  Este estilo agrupa los datos y los mecanismos necesarios para manipularlos. Un subconjunto de estos mecanismos se pone a disposición del resto de componentes como si fueran servicios públicos.

  Este estilo incrementa la cambiabilidad y también la capacidad del sistema para ser reutilizable.

  Un ejemplo de la representación gráfica del estilo Orientado a objetos es el siguiente dibujo:



- ***Por capas***:

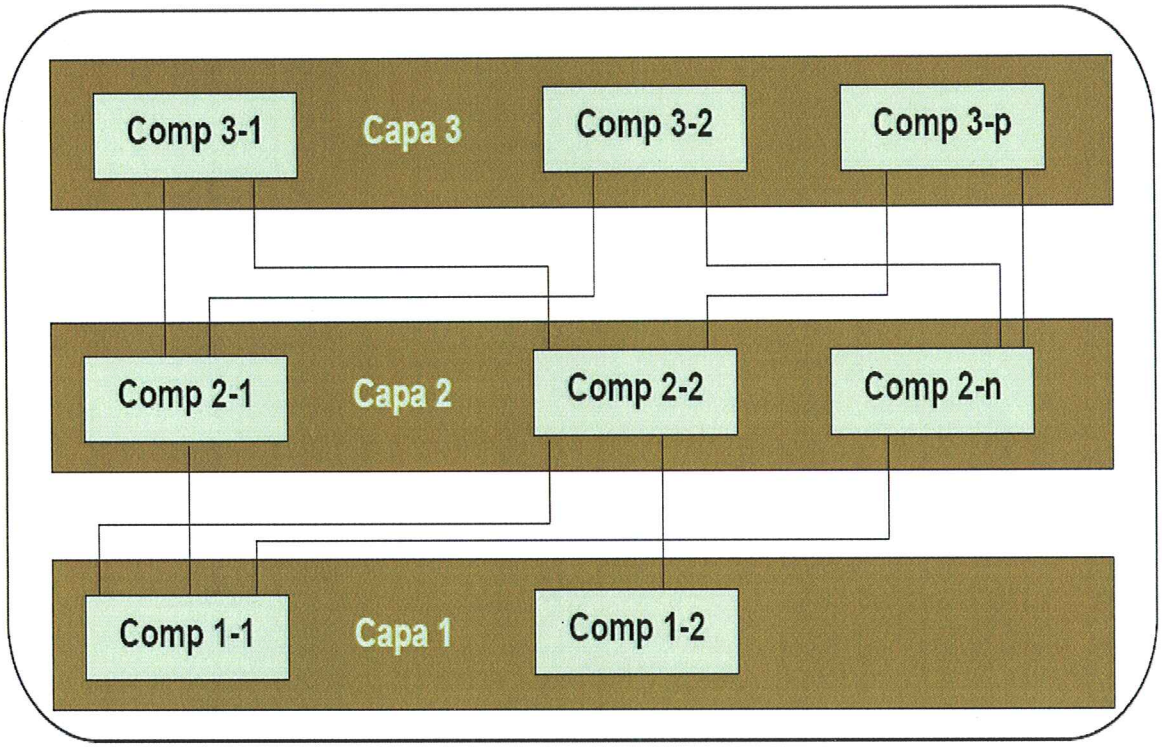
  En este estilo los componentes de la arquitectura software se organizan en forma de capas.

  Todos los componentes de la arquitectura se distribuyen en las diferentes capas, de tal manera que dos componentes pueden comunicarse solamente en el caso que ambos estén alojados en la misma capa, o bien estén alojados en capas contiguas.

  Este estilo incrementa la cambiabilidad y también la portabilidad del sistema.

  Pero también es necesario destacar que éste estilo desfavorece la capacidad de eficiencia del sistema.

Un ejemplo de la representación gráfica del estilo por capas es el siguiente dibujo:



- ***Programa principal y subrutina***:

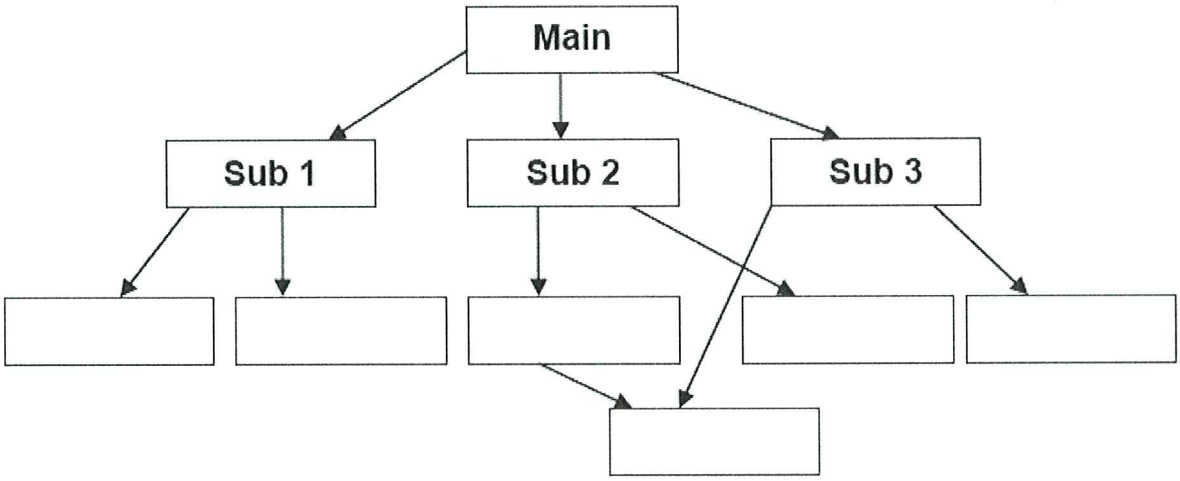
  Este tipo de patrón, es el tradicional estilo de programación resultado de un análisis y un diseño descendente.

  Los componentes de este estilo arquitectónico se agrupan en una jerarquía organizada en programa principal y subrutinas.

  Una subrutina es un componente que cuando recibe el control y los datos de uno de sus padres, lo transmite a todos sus hijos. En cambio cuando una subrutina devuelve el control, lo hace en sentido contrario, es decir se propaga de hijos a padres.

  Debido a ésta jerarquía organizada, y a como interactúan sus componentes, podemos afirmar que este tipo de patrón arquitectónico mejora la propiedad de cambiabilidad del sistema y en cambio afecta negativamente a la propiedad de eficiencia del sistema.

Un ejemplo de la representación gráfica del estilo programa principal y subrutina es el siguiente dibujo:



- ***Concentrado en los datos***

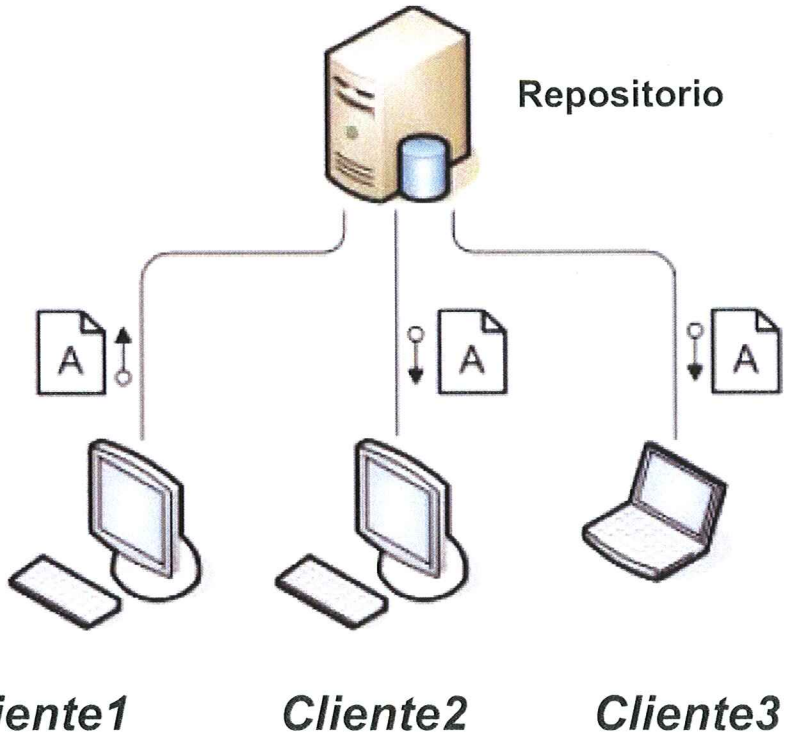
Este tipo de patrón, es una arquitectura útil en sistemas que necesitan un acceso a los datos que son compartidos. Estos datos compartidos se almacenan en un mismo lugar, como un repositorio.

Los clientes se conectan a este repositorio para poder consultar y modificar los datos.

También existe una variante sobre este tipo de patrón, en el que se substituye el repositorio, que es un medio de almacenaje pasivo, por un repositorio activo o pizarra.

La diferencia principal entre estas dos variantes del patrón es que en el repositorio activo o pizarra se notifican los cambios de los datos a los clientes que están interesados en estos datos que se han modificado.

Un ejemplo de la representación gráfica del estilo concentrado en los datos es el siguiente dibujo:



Tal y como vemos en el dibujo mostrado, hay unos componentes llamados clientes.

Los clientes son procesos independientes, que pueden cambiar de plataforma o cambiar su implementación, sin que estos cambios tengan que afectar al resto de clientes. También sería posible que se añadieran nuevos clientes sin que la arquitectura general del sistema se viera afectada. Por estas razones podemos afirmar que este tipo de estilo arquitectónico favorece las propiedades de cambiabilidad y la interoperabilidad del sistema.

- ***Flujo de datos***

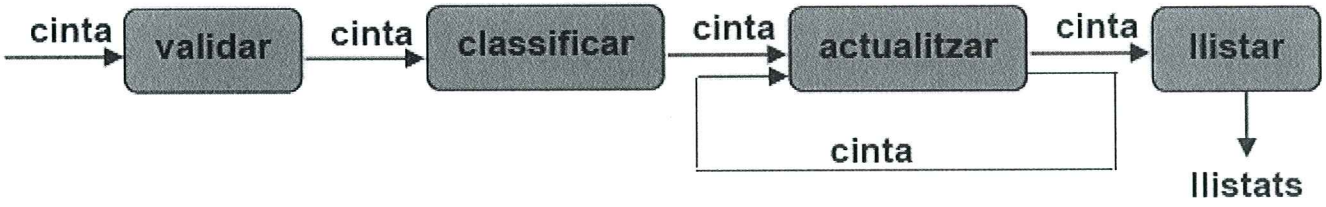
Este tipo de patrón, es un estilo que se suele estar enfocado a sistemas que para unos determinados datos de entrada , también llamados input, realizan una serie de transformaciones para poder obtener unos datos de salida, también llamados output.

Dentro de este estilo arquitectónico se pueden diferenciar dos tipos de patrones de flujo de datos:

- ***Batch secuencial***:

Este tipo de patrón organiza el sistema en una secuencia de programas independientes, que se intercambian datos entre ellos. Un programa no puede comenzar su ejecución hasta que todos los programas anteriores hayan finalizado su ejecución. Este tipo de patrón se aplica normalmente en procesos que se ejecutan de una forma diferida o batch.

Un ejemplo de la representación gráfica del estilo Batch secuencial es el siguiente dibujo:

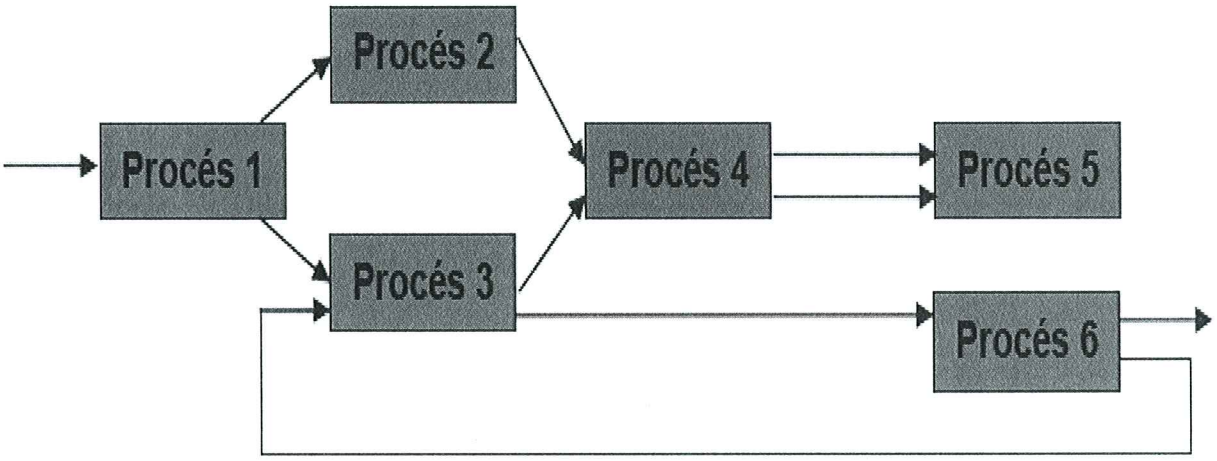


- ***Tubos y filtros***:

Este tipo de patrón se estructura en un grupo de procesos o subsistemas llamados filtros. Estos filtros están conectados por unos tubos, que transmiten los datos entre dos componentes del sistema.

Todos los filtros están diseñados para recibir una entrada de datos, o input, de una cierta manera, y producir una salida de datos, u output, de otra manera.

Un ejemplo de la representación gráfica del estilo tubos y filtros es el siguiente dibujo:



5.1.2. ARQUITECTURA SOFTWARE ESCOGIDA

Después de haber analizado, los distintos tipos de patrones arquitectónicos en el apartado anterior, podemos deducir que no es posible encontrar un patrón que mejore la eficiencia del sistema y también mejore la cambiabilidad y la portabilidad del sistema.

Esto es debido a que el criterio de eficiencia está reñido con los criterios de cambiabilidad y portabilidad, es decir cuanto más cambiabilidad y más portabilidad tiene un sistema menos eficiente es.

Garantizar una mayor cambiabilidad del sistema, supone agrupar y organizar los datos y las funciones en clases, módulos o capas de abstracción que, de alguna manera, hagan que la implementación de los componentes del sistema no sea transparente.

Como consecuencia de la "*no transparencia*" de los componentes, se obtiene una mayor complejidad en la implementación del sistema, cosa que afecta negativamente a la eficiencia del sistema, ya que para realizar una misma tarea se deben consultar a diferentes componentes o atravesar diferentes capas de software.

En consecuencia al problema que nos encontramos entre el criterio de eficiencia y el de portabilidad y cambiabilidad, es aconsejable llegar a un equilibrio, con el fin de favorecer las propiedades de cambiabilidad y portabilidad del sistema, lo máximo posible, sin provocar que la eficiencia del sistema se vea muy desfavorecida.

Para poder tomar una decisión acerca de escoger el patrón adecuado, hemos tenido en cuenta los siguientes factores:

- La problemática existente a la hora de mejorar la eficiencia, la portabilidad y la cambiabilidad el sistema.
- La necesidad de encontrar un equilibrio entré las propiedades de cambiabilidad, portabilidad y eficiencia,

Después de haber tenido en cuenta los factores anteriores, podemos afirmar que de todos los patrones analizados en el apartado anterior, el que más se ajusta a las necesidades del editor visual de formularios vía web, es el patrón arquitectónico en Capas, y más concretamente en 3 capas.

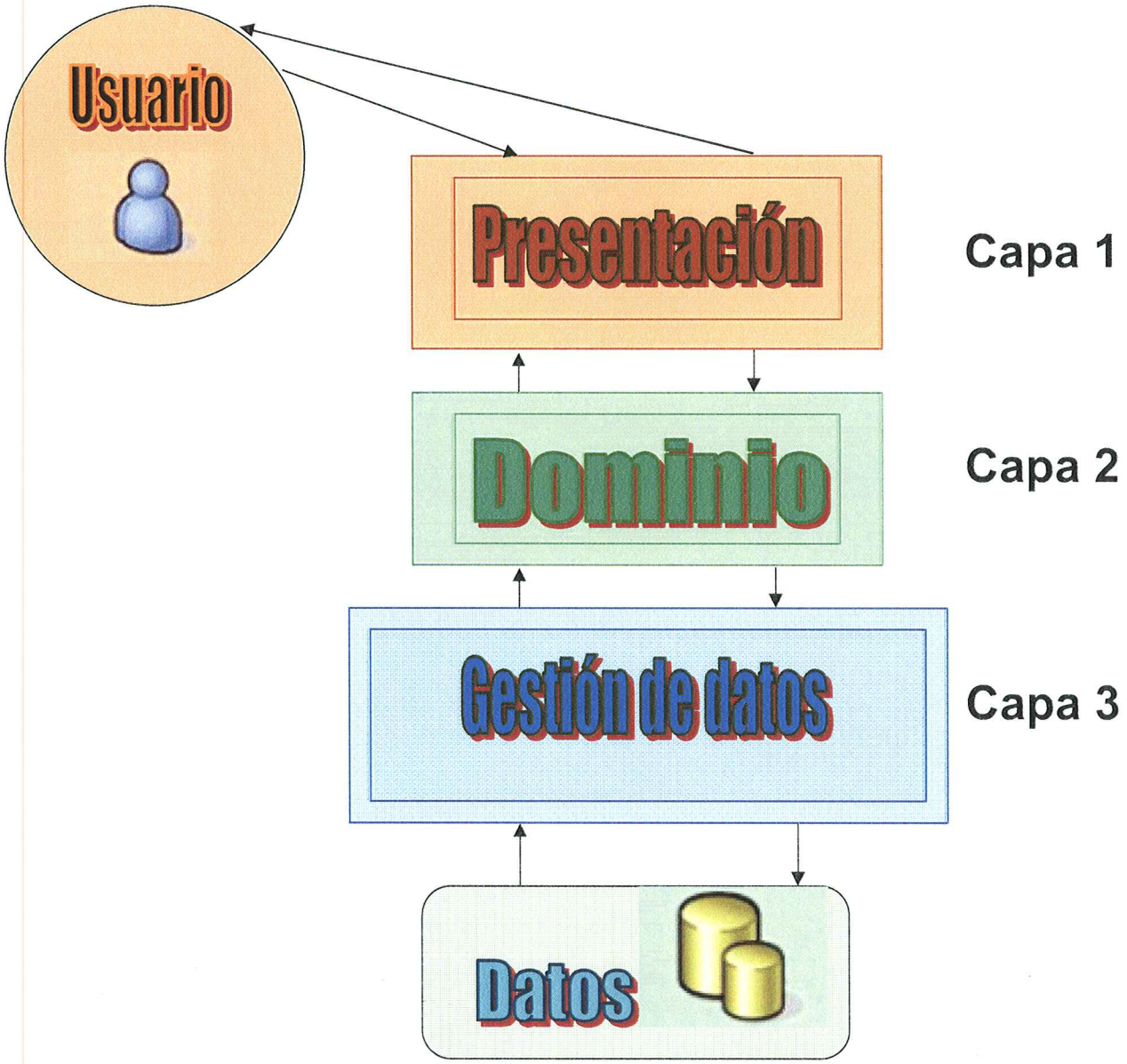
Pero además de utilizar el patrón en Capas, también se ha considerado adecuado utilizar patrón de orientación a objetos, ya que el lenguaje utilizado para implementar el editor visual de formularios vía web, es orientado a objetos.

5.1.3. PATRÓN EN CAPAS

Uno de los objetivos principales del patrón en Capas es separar el código de la aplicación del código de la presentación. Cada capa puede acceder únicamente otra capa que se encuentre en un nivel contiguo.

El diseño de una aplicación que utilice una arquitectura de capas presenta la gran ventaja de que producirá código modular en el que la modificación de uno de sus componentes no exige modificar otro de los componentes pertenecientes a otra capa.

El diseño en capas, también favorece la posible reutilización de código para futuras aplicaciones. En la siguiente figura se muestra la estructura en 3 capas de nuestro sistema:



5.1.4. LA ORIENTACIÓN A OBJETOS

La orientación a objetos más que un patrón arquitectónico a seguir, es una metodología de diseño.

En la orientación a objetos se define, lo que en términos de programación se llama clase de objetos. La idea principal de la orientación a objetos es subir el nivel de programación al nivel de objetos, ya que un objeto es algo mucho más intuitivo y comprensible por el ser humano, que los términos que se utilizan normalmente en programación como bytes, bits, registros etc...

Un objeto es un paquete de datos del sistema que tiene una serie de atributos y de métodos, que pueden utilizar los datos que contiene el objeto.

Los atributos son las operaciones más relevantes del objeto y los métodos son los instrumentos necesarios para manipular estos atributos.

Las clases de objetos definen como son los objetos, es decir, los tipos de datos que guardarán y los métodos que tendrán.

La programación orientada a objetos, estructura un programa como un conjunto de objetos, que se pueden comunicar entre ellos para realizar tareas.

La orientación a objetos ayuda a hacer programas que tengan módulos más fáciles de construir, de mantener y de reutilizar.

Las características más importantes de la orientación a objetos son las siguientes:

- ***Ocultación de la información.***

  Esta característica garantiza que los objetos no puedan cambiar el estado interno de otros objetos de maneras inesperadas, ya que únicamente los propios objetos tienen métodos internos al objeto que permiten acceder y modificar su estado.

  Para controlar el posible cambio de estado desde otros objetos, cada objeto ofrece una interfaz a los demás objetos, que especifica los métodos que pueden usar el resto de objetos para poder interactuar con el.

  A pesar del posible riesgo que comporta, hay algunos lenguajes orientados a objetos que permiten un acceso directo a los datos internos del objeto, pero de una manera controlada y limitando el grado de abstracción.

- ***Abstracción.***

  Cada objeto en el sistema sirve como un agente abstracto, que puede realizar tareas, informar, cambiar de estado, y comunicarse con otros objetos del sistema, sin la necesidad de que el resto de objetos conozcan como se implementan sus métodos.

- ***Herencia.***

La herencia es un tipo especial de relación entre clases de objetos. El mecanismo de la herencia bien utilizado, facilita la capacidad de modificar y reutilizar los diseños y el código.

La herencia consiste en 3 conceptos fundamentales:

- Existen dos tipos de clases, a las que llamaremos padre (Superclase o clase base) e hija (subclase o clase Derivada).
- Al igual que las herencias en la vida real, la clase hija pasa a tener lo que tiene la clase padre. Así que heredará métodos y atributos.
- Un objeto de la clase hija es también un objeto de la clase padre.

En la clase hija se definen las diferencias respecto de la clase padre. La herencia se puede usar con el objetivo de extender las funcionalidades de la clase padre y para especializar su comportamiento

- ***Polimorfismo.***

El concepto de polimorfismo tiene sentido gracias al concepto de herencia. Esto es debido a que gracias a la herencia, se puede interpretar que un objeto de una subclase es también un objeto de una superclase.

El polimorfismo es un mecanismo que se aprovecha de la herencia (especialmente de interfaz) para manejar indistintamente objetos de las subclases como si fuesen objetos de la clase base, sin preocuparse por la clase en concreto a l a que pertenecen.

El polimorfismo es interesante utilizarlo cuando un comportamiento varía en función del tipo del objeto.

Teniendo en cuenta las principales características de la orientación objetos, que se han mostrado anteriormente, pasaremos a listar las ventajas e inconvenientes que implica aplicar este tipo de patrón.

- **Ventajas**:

1. El coste de la implementación es menor que, aplicando otro tipo de patrones.

2. Permite dividir las responsabilidades más fácilmente dentro de un mismo grupo de trabajo.
3. Aumenta la propiedad de reutilización del software.

4. Los sistemas que utilizan el patrón de orientación a objetos resultan más viables económicamente a medio y a largo plazo.

5. Permite modelar de una manera más fiable y eficaz los dominios.

6. Facilita el análisis y el diseño de la aplicación.

7. Las aplicaciones desarrolladas con el patrón de orientación a objetos, ofrecen una mayor seguridad gracias a la ocultación de los datos y de la implementación interna de las clases.

8. Aumenta la propiedad de extensibilidad, gracias a la herencia y al permitir añadir nuevos módulos.

- **Inconvenientes:**

1. A pesar que es una técnica que se aprende rápidamente, es muy difícil de dominar completamente.
2. Aumenta los costes de desarrollo a corto plazo
3. Aumenta el tiempo mínimo necesario para poder ver una primera versión del sistema a desarrollar, en funcionamiento.
4. Disminuye en cierto grado la eficiencia del sistema.

Después de analizar tanto el dominio editor visual de formularios vía web como todo lo referente al patrón de orientación a objetos, valorando los inconvenientes y las ventajas que ofrece, se ha decidido que es apropiado aplicar el patrón de orientación a objetos para desarrollar el editor visual de formularios vía web.

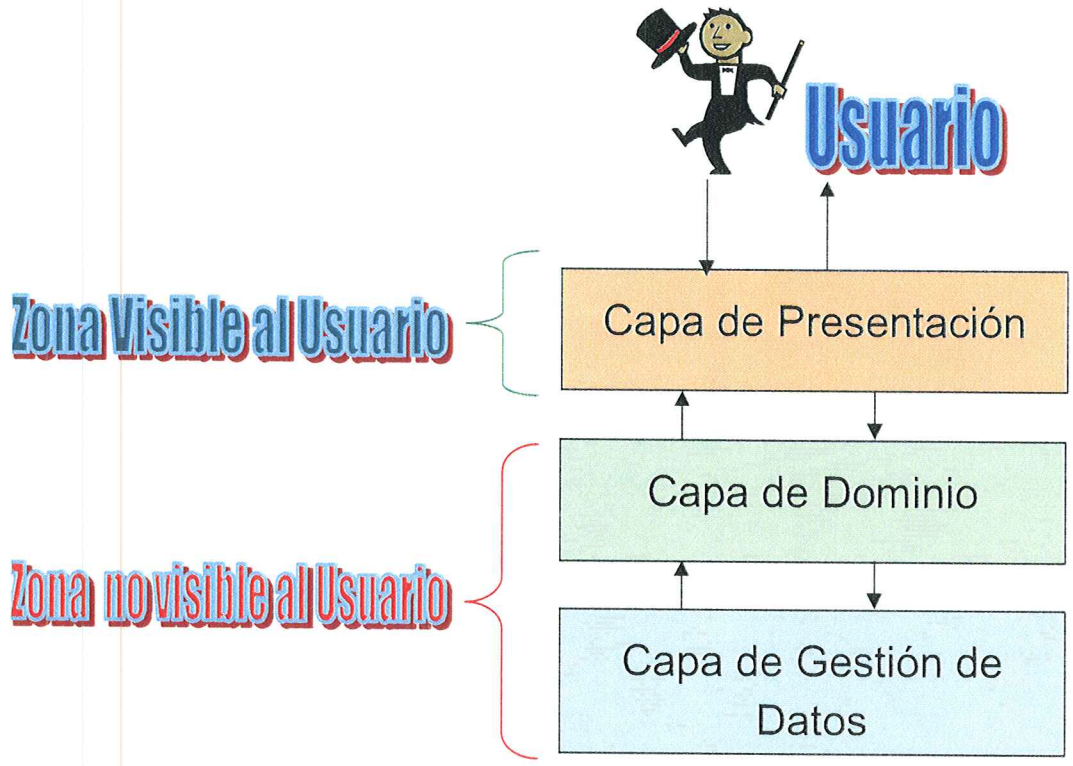
5.2. Capa de presentación

Como se ha explicado en el apartado anterior, se ha tomado la decisión de utilizar el patrón arquitectónico en Capas, y más concretamente en 3 capas, para construir el editor visual de formularios vía web.

La primera de las 3 capas, en que estructuraremos nuestro sistema software, es la capa de presentación.

La capa de presentación es la capa responsable de interactuar con el usuario y se corresponde con lo que tradicionalmente se llama la interfaz de usuario. La capa de presentación esta guiada por los eventos que produce el usuario, cuando desea interactuar con el sistema software que se va a desarrollar.

La capa de presentación es la única capa que ve el usuario, es la única capa que el sistema presenta al usuario. Por este motivo, el usuario únicamente tendrá acceso a la capa de presentación.



Como puede apreciarse en la figura anterior, la única capa con la que puede interactuar el usuario es la capa de presentación. De este modo, la capa de presentación presenta información al usuario y captura la información que el usuario introduce.

Con la información que recoge la capa de presentación del usuario, realiza un pequeño tratamiento de los datos, para comprobar que no hay errores de formato, y una vez depurada esta información, la capa de presentación la comunica a la capa contigua, es decir le pasa la información a la capa de dominio.