PATRONES USADOS EN DRUPAL

Control de Versiones

|  |  |  |  |
| --- | --- | --- | --- |
| **Modificación** | **Fecha** | **Autor** | **Versión** |
| Versión Inicial. | 26/05/2014 | Alba Moragrega | 1.0 |
|  |  |  |  |

Índice y Contenidos

[**1.** **ARQUITECTURA DE DRUPAL** 4](#_Toc389393901)

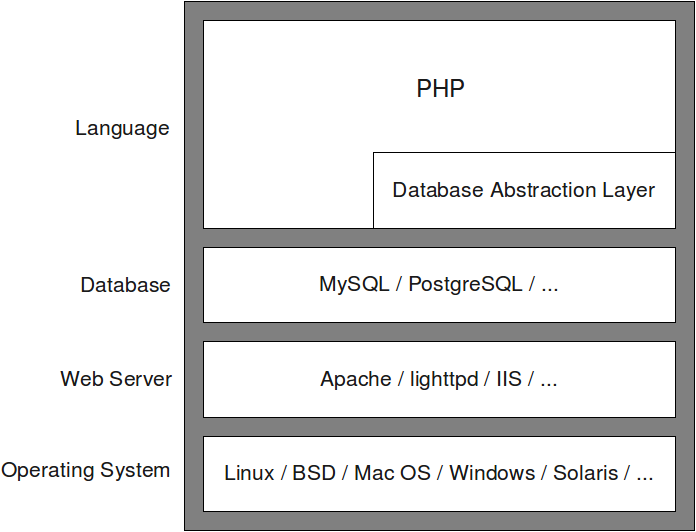
[**2.** **DESCRIPCIÓN DEL SOFTWARE** 5](#_Toc389393902)

[2.1 Software utilizado para el desarrollo Android 5](#_Toc389393903)

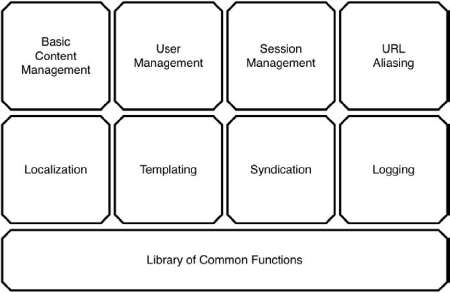
[2.2 Software utilizado para el desarrollo Web 5](#_Toc389393904)

# **ARQUITECTURA DE DRUPAL**

Drupal está diseñado para funcionar sobre prácticamente cualquier entorno independientemente del sistema operativo, servidor web o sistema de gestión de base de datos.



Además Drupal proporciona un framework responsable de proveer las funcionalidades básicas necesarias en las otras partes del sistema, así como de servir de soporte para las nuevas funcionalidades.



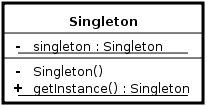
El framework que Drupal proporciona facilita la ampliación de nuevas funcionalidades a partir de unidades autocontenidas denominadas “módulos”. A través de dichos módulos Drupal marca el camino que los desarrolladores han de seguir para ampliarlo funcionalmente, ya sea creando nuevos tipos de contenidos como noticias, eventos, etc. o con nuevas funcionalidades como, por ejemplo, autenticación de usuarios frente a LDAP, integración con el gestor documental Drupal, soporte para e-commerce, etc.

# **PATRONES DE DISEÑO EN DRUPAL**

Dentro del núcleo de Drupal se hace uso de multitud de patrones de diseño, entre ellos podríamos citar los siguientes:

# Singleton

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).

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

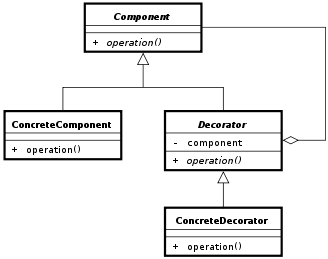
El patrón singleton provee una única instancia global gracias a que:

* La propia clase es responsable de crear la única instancia.
* Permite el acceso global a dicha instancia mediante un método de clase.
* Declara el constructor de clase como privado para que no sea instanciable directamente.

Dentro del núcleo de Drupal se utiliza este patrón de diseño en diversas tareas como la gestión del pool de conexiones con la base de datos y pensando en los módulos y temas de Drupal como objetos para llevar a cabo la gestión de dichos elementos.

# Decorator

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.

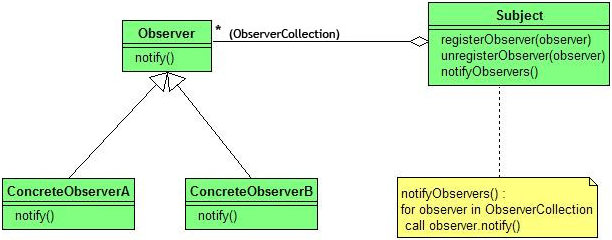


Este patrón de diseño es utilizado en el núcleo de Drupal para facilitar la extensión de sus componentes.

# Observer

El patrón observer define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador se encarga de notificar este cambio a todos los otros dependientes.

El objetivo de este patrón es desacoplar la clase de los objetos clientes del objeto, aumentando la modularidad del lenguaje, así como evitar bucles de actualización (espera activa o polling).

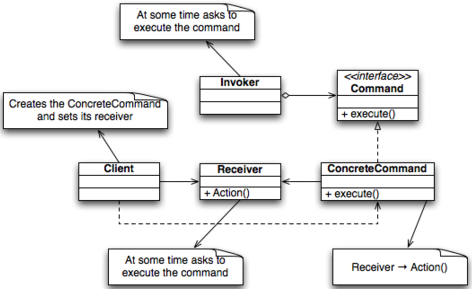


Al igual que en el caso del patrón de diseño Decorator este módulo es utilizado para llevar a cabo la extensión de los componentes internos de Drupal a través de los correspondientes hooks.

# Command

Este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además se facilita la parametrización de los métodos:

* Encapsula un mensaje como un objeto, con lo que permite gestionar colas o registro de mensaje y deshacer operaciones.
* Soportar restaurar el estado a partir de un momento dado.
* Ofrecer una interfaz común que permita invocar las acciones de forma uniforme y extender el sistema con nuevas acciones de forma más sencilla.

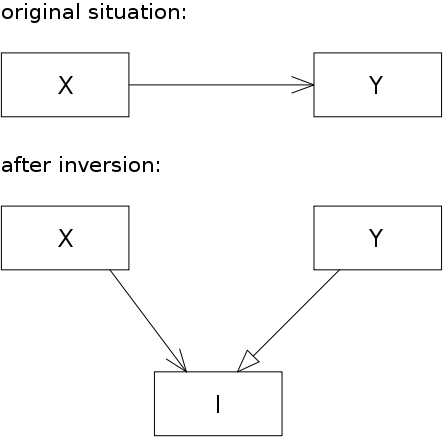


Este patrón de diseño es utilizado en Drupal para permitir llevar a cabo la ejecución de ciertas tareas pasando como parámetro el operador, esta es la base fundamental del funcionamiento de los hooks.

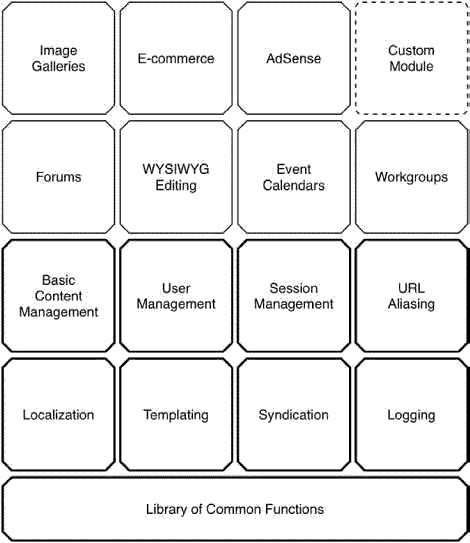
# Módulos

Drupal proporciona los módulos para extender su funcionalidad. La funcionalidad que proporcionan los módulos puede ser habilitada o deshabilitada a través de la correspondiente página de administración.

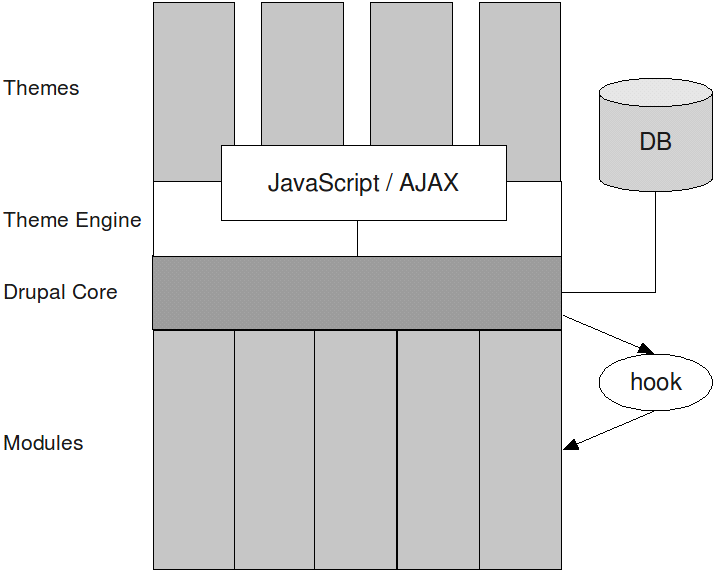
Drupal consigue proveer esta funcionalidad gracias a la implementación que realiza del patrón de diseño “**Inversion of Control**”, este patrón de diseño consiste en un cambio en el flujo de ejecución de un programa en el que en vez de especificar el flujo de la información se especifica la respuesta esperada de una operación.



De esta forma los nuevos módulos que se añaden a Drupal se disponen de la siguiente forma dentro del core de Drupal.



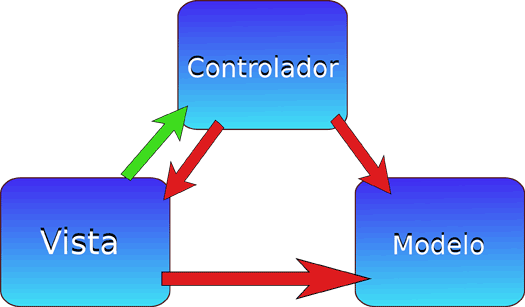
Así mismo Drupal proporciona “themes” que permiten la personalización de la apariencia del sitio web para adaptarlo a la entidad corporativa de la empresa con la que se está trabajando, de esta forma se consigue obtener un portal totalmente adecuado y personalizado a las necesidades del cliente.



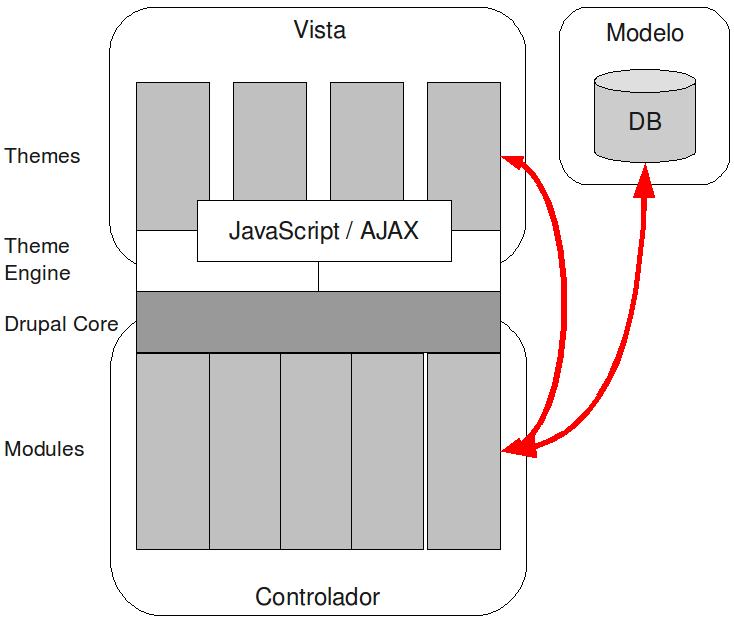
# Drupal y MVC

El **modelo-vista-controlador** (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el SGBD y el controlador es el responsable de recibir los eventos de entrada desde la vista:

* **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.
* **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
* **Controlador**: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.



En Drupal este patrón arquitectónico está implementado de la siguiente forma:



Como se puede observar en la utilización que Drupal hace del patrón arquitectónico MVC no existe interacción entre la vista y el modelo, esto se debe a que dichas interacciones siempre se realizan a través de la lógica de negocio, el controlador.