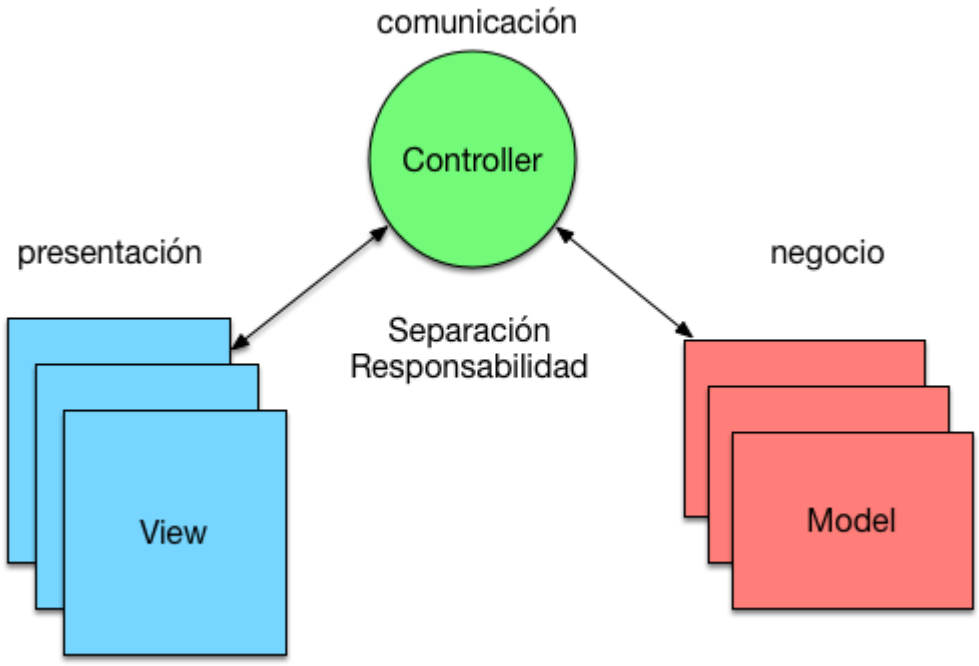
Uno de los patrones de diseño más utilizados hoy en día es el patrón MVC. La mayoría de los frameworks web se apoyan en él desde Spring MVC , ASP.NET MVC pasando por Laravel. El patrón MVC se basa en la división de responsabilidades.

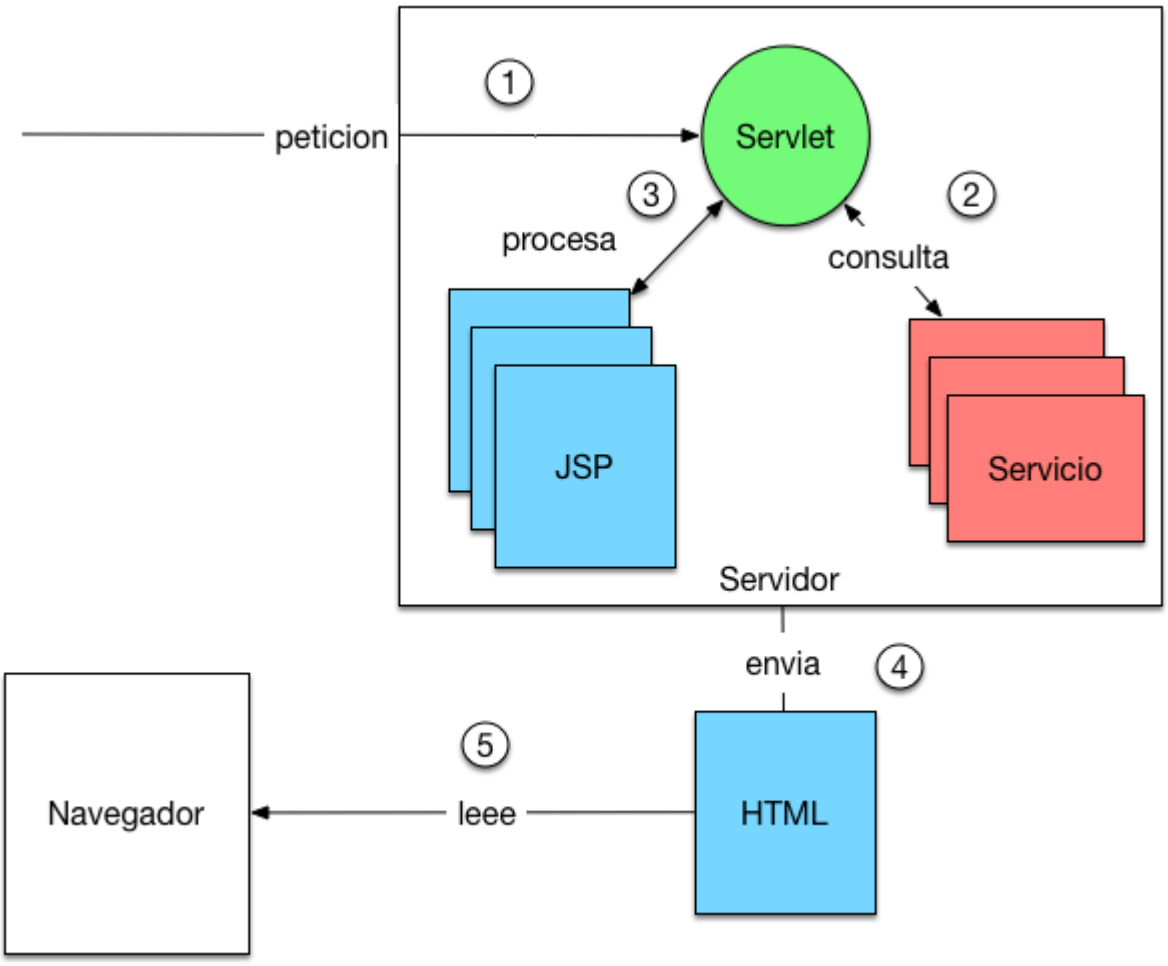


En este patrón existen tres responsabilidades. La primera de ellas es la responsabilidad del Modelo que se encarga la mayor parte de la lógica de negocio y comunicación con las bases de datos. En segundo lugar tenemos la Vista que se encarga de mostrar la información del modelo y por último el Controlador que comunica la información entre ambas partes.

El patrón MVC en el servidor

Durante la última década ha sido muy habitual desarrollar aplicaciones web utilizando el

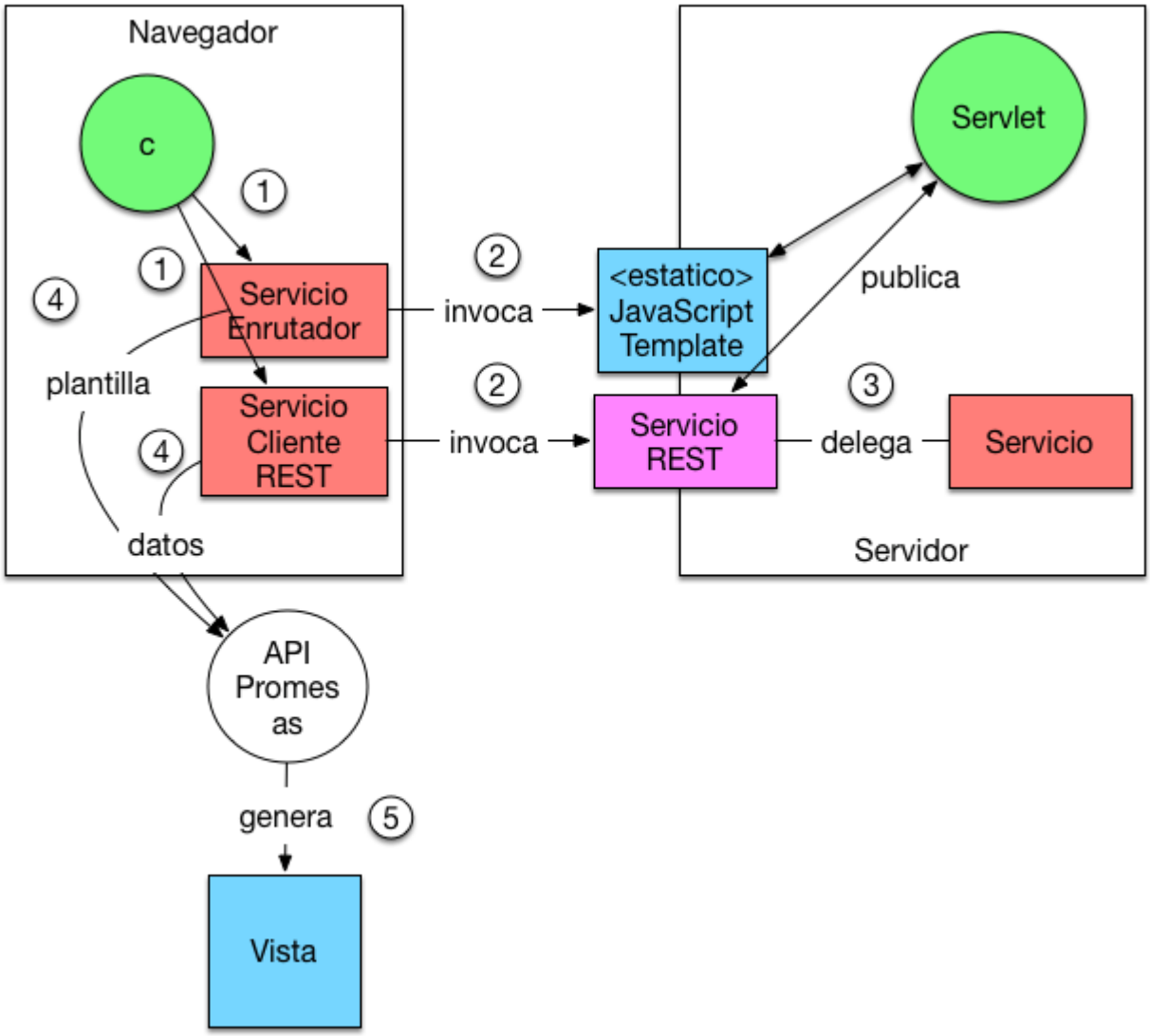
patrón MVC del lado del servidor , en el mundo Java se ha usado ampliamente.



Este diagrama nos muestra de forma simplificada como funciona el patrón MVC en una aplicación Java EE clásica. El Servlet hace la función de controlador , las clases de Servicio hacen la función de modelo junto con los Bussiness Objects. Por último los ficheros JSP o algún motor de plantillas se encargan de la Vista. El resultado final de todas estas operaciones es generar un documento HTML que el navegador presenta.

El patrón MVC en el cliente

Las cosas están cambiando y las nuevas arquitecturas SPA (Simple Page Application) están moviendo este patrón de diseño hacia el lado cliente , eliminando parte de la funcionalidad del servidor.



Aquí las cosas cambian de forma importante . La mayor parte de las responsabilidades pasan a ser del Cliente y quedan pocas en el lado Servidor. El servidor se encargará de publicar plantillas de JavaScript y datos en formato JSON o similar. El cliente construido en JavaScript usará un enrutador y un API de promesas o similar para fundir las plantillas y los datos que vienen del servidor a través de peticiones asíncronas. Este tipo de arquitecturas todavía esta en plena evolución pero tiene algunas ventajas claras.

Al favorecer la creación de servicios REST , favorecemos la integración de otras aplicaciones que consuman estos servicios. Abrimos un poco más la puerta a cambiar la plataforma del Servidor ya que sus responsabilidades se han reducido. Por último apostamos de forma más fuerte por la programación asíncrona algo que cada día es más importante ya que nuestras aplicaciones tienden a comunicarse con otras muchas.

Otros artículos relacionados : Patrón Factory , Patrón Flyweight