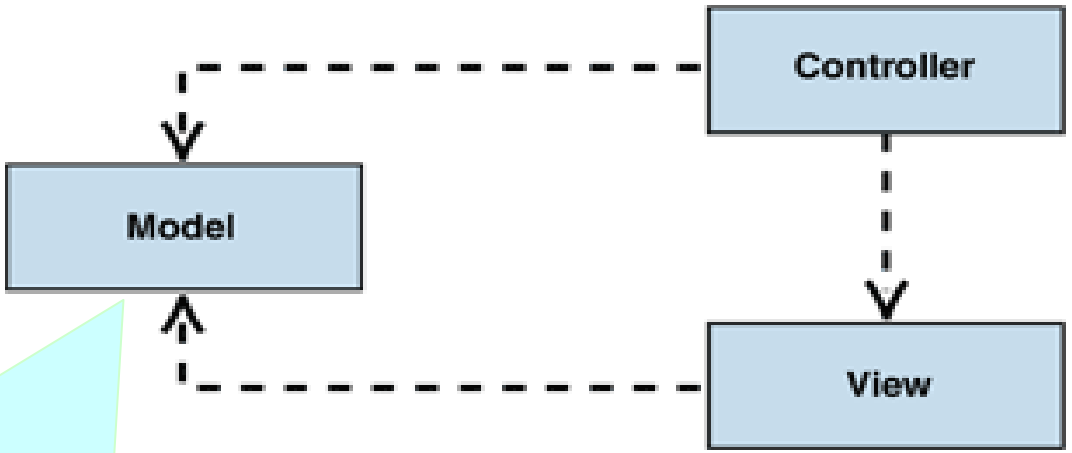
Patron de Arquitectura de Software MVC

Modelo-Vista-Control

Descripción del patrón

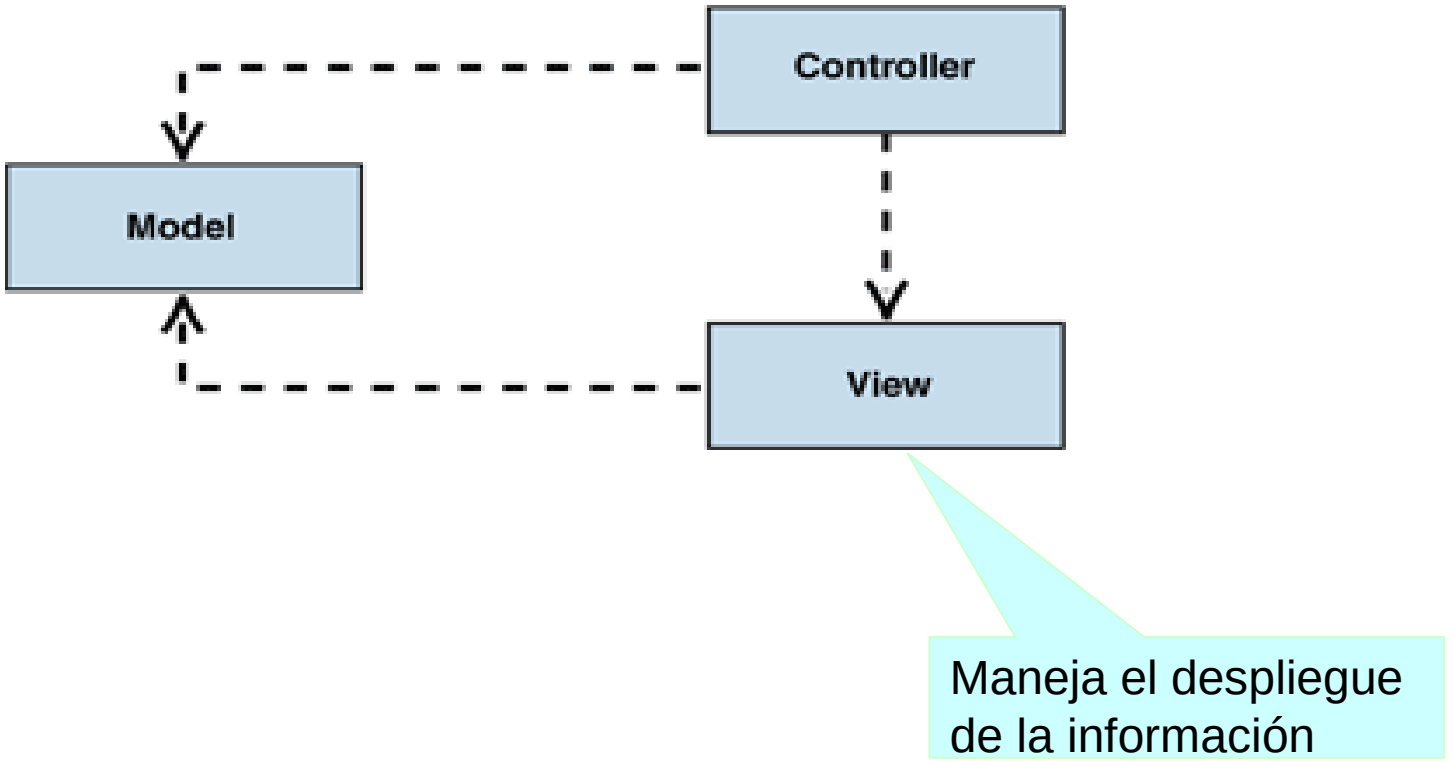
- Problema: Como modularizar la funcionalidad de la interfaz de usuario de una aplicación Web de tal forma que usted pueda modificar fácilmente sus partes individuales?
- Solución: El patrón MVC (Model-View-Controller) separa el modelado del dominio, la presentación y las acciones basados en las entradas del usuario en tres clases apartes. [Burbeck92]

Vista del patrón



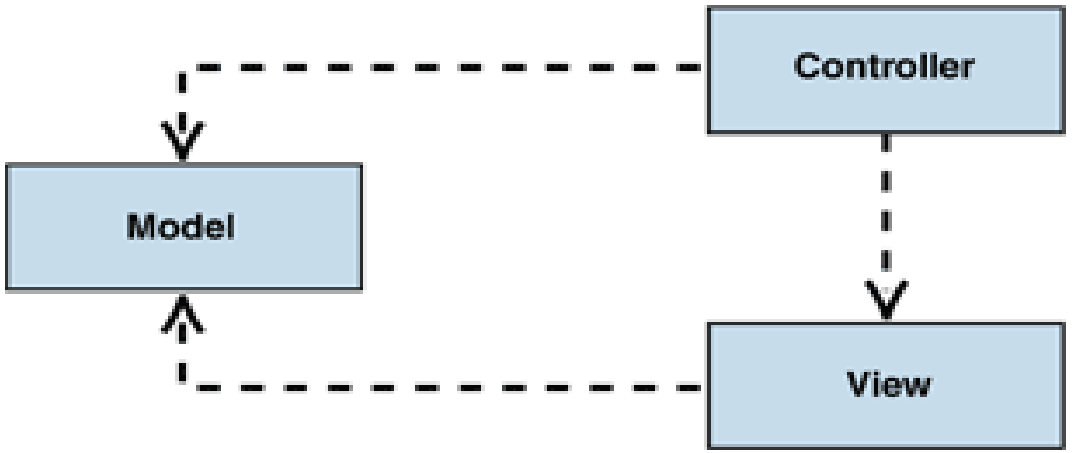
Maneja el comportamiento y los datos del dominio de la aplicación, responde a los requerimientos de información acerca de su estado (usualmente desde la vista) y responde a las instrucciones para cambiar de estado (usualmente desde el controlador)

Vista del patrón

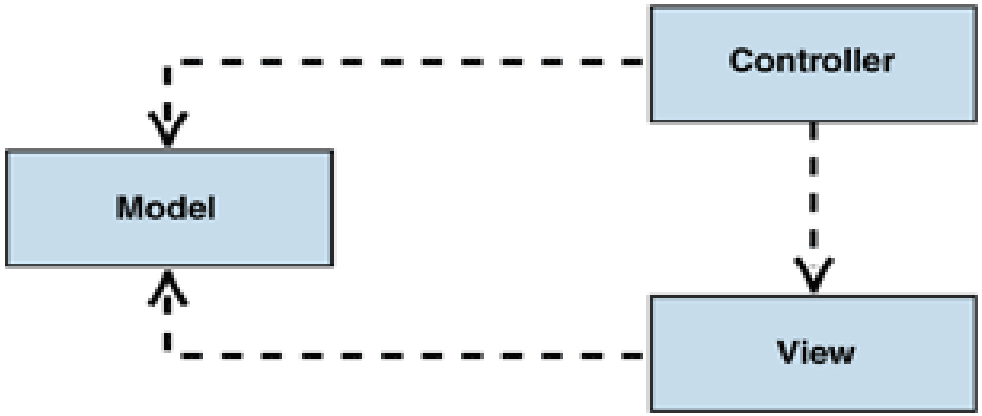


Vista del patrón

Interpreta las acciones del usuario de teclado y ratón, informando al modelo y/o a la vista para cambiar apropiadamente sus estados.

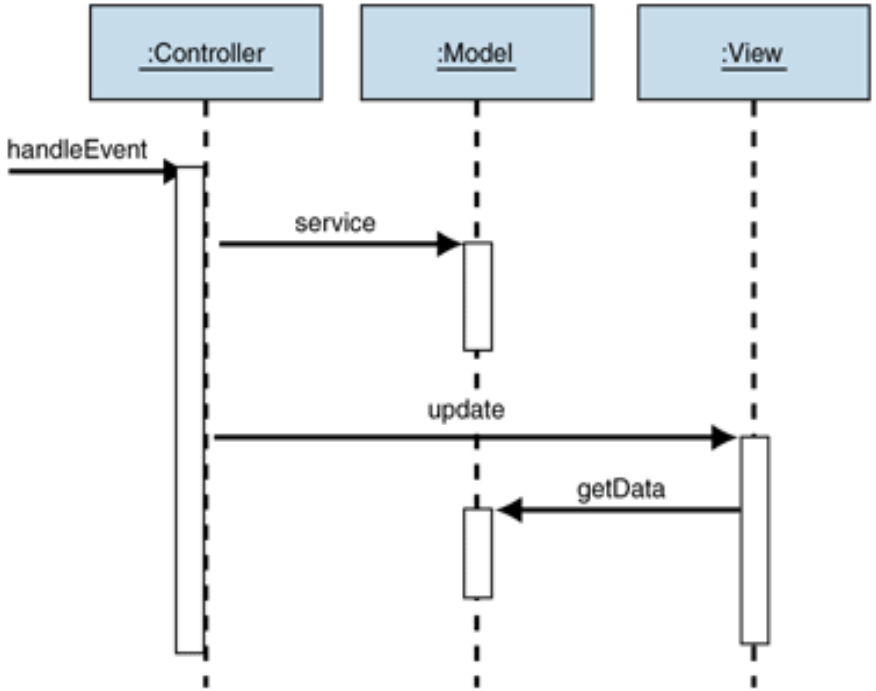


Dependencias



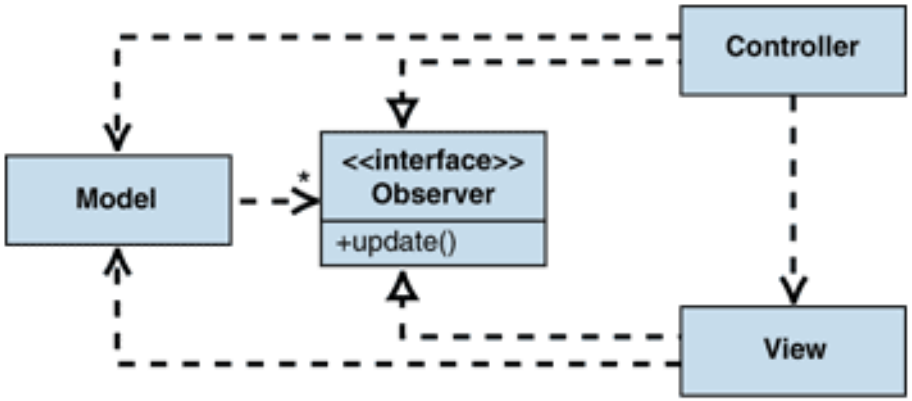
- Tanto la vista como el controlador dependen del modelo. Sin embargo, el modelo no depende ni de la vista ni del controlador.
- La separación permite que el modelo sea construido y probado independientemente de la presentación visual.
- La separación entre vista y controlador es secundaria en muchas aplicaciones, sin embargo en las aplicaciones Web la vista (el navegador) y el controlador (los componentes del lado servidor) están bien definidos.

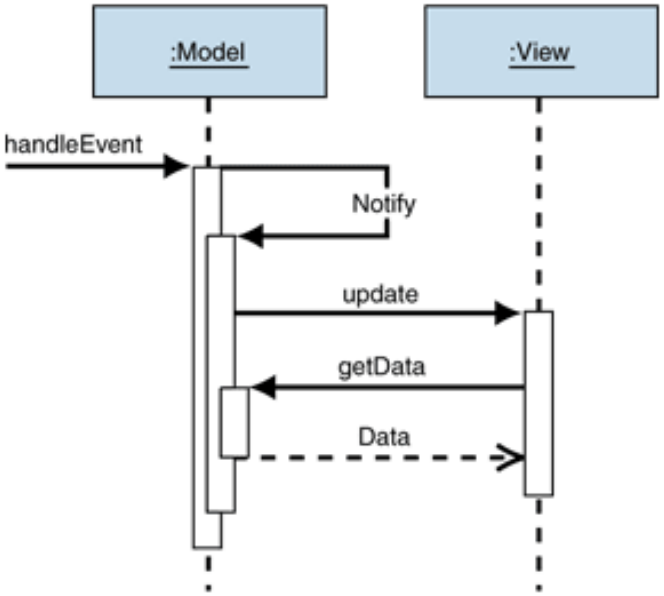
Comportamiento MVC (pasivo)



•Es utilizado cuando un controlador manipula el modelo exclusivamente
•El controlador modifica el modelo y le informa a la vista que este ha cambiado y debe ser refrescada.
•En este escenario el modelo es completamente independiente de la vista y del controlador

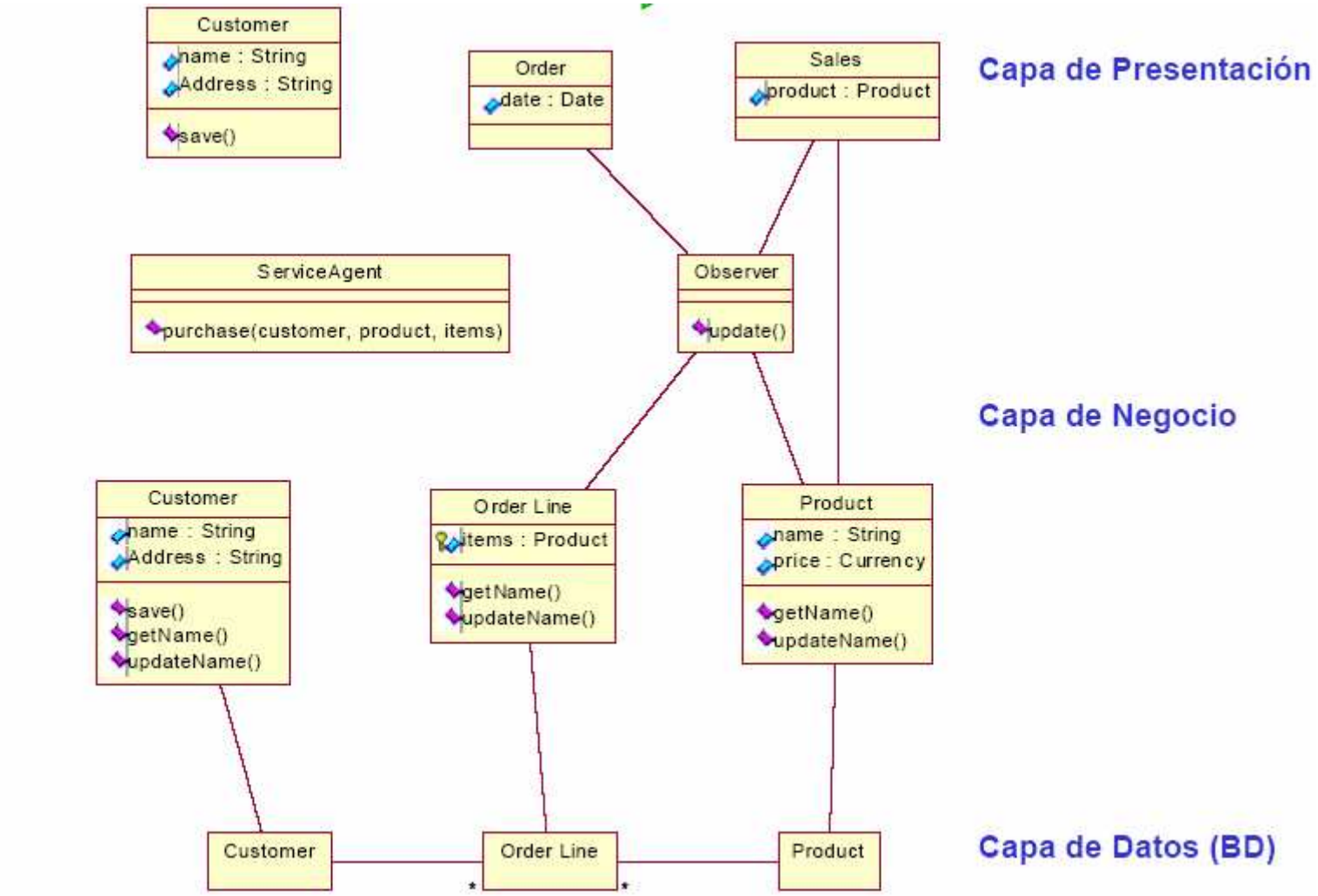
Comportamiento MVC (activo)



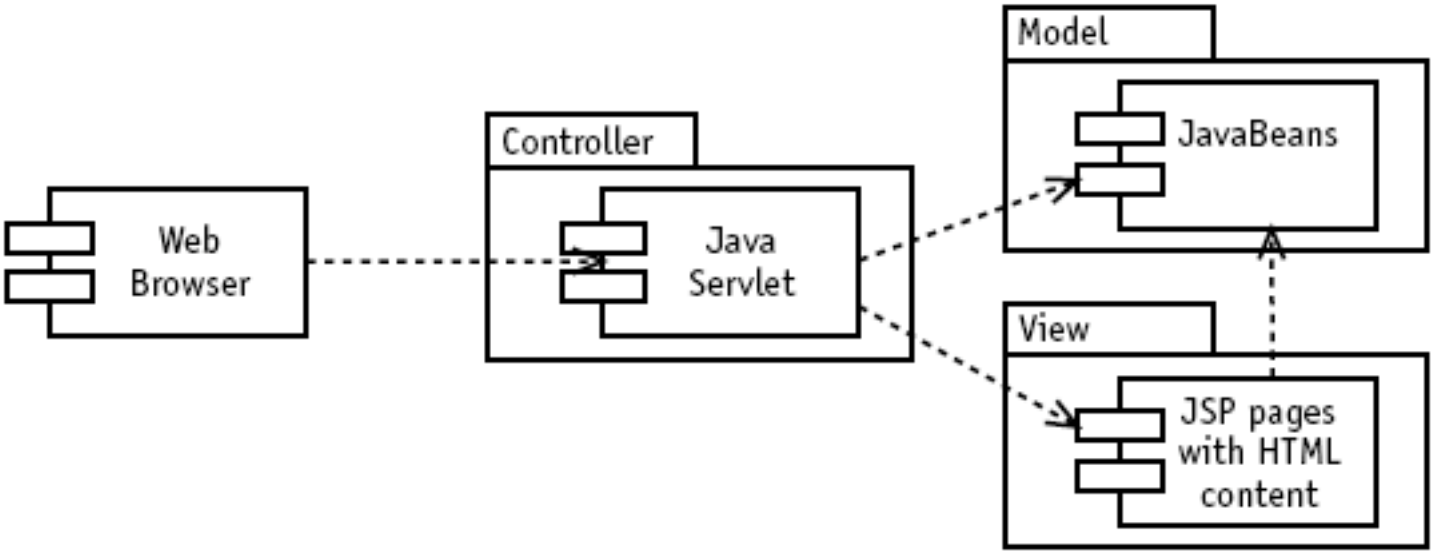


•Es usado cuando el modelo cambia de estado sin la intervención del controlador, lo cual puede pasar cuando otras fuentes están cambiando los datos y los datos deben reflejarse en la vista.
•Debido a que solo el modelo detecta los cambios a su estado interno cuando estos ocurren, el modelo deberá notificar a la vista para refrescarla, pero esto crearía una dependencia entre el modelo y la vista, lo cual iría en contra de uno de los principios del patrón *MVC.*
•Como solución, se introduce el patrón *Observer*, el cual provee un mecanismo para alertar a otros objetos de cambios de estado sin introducir dependencias entre ellos.

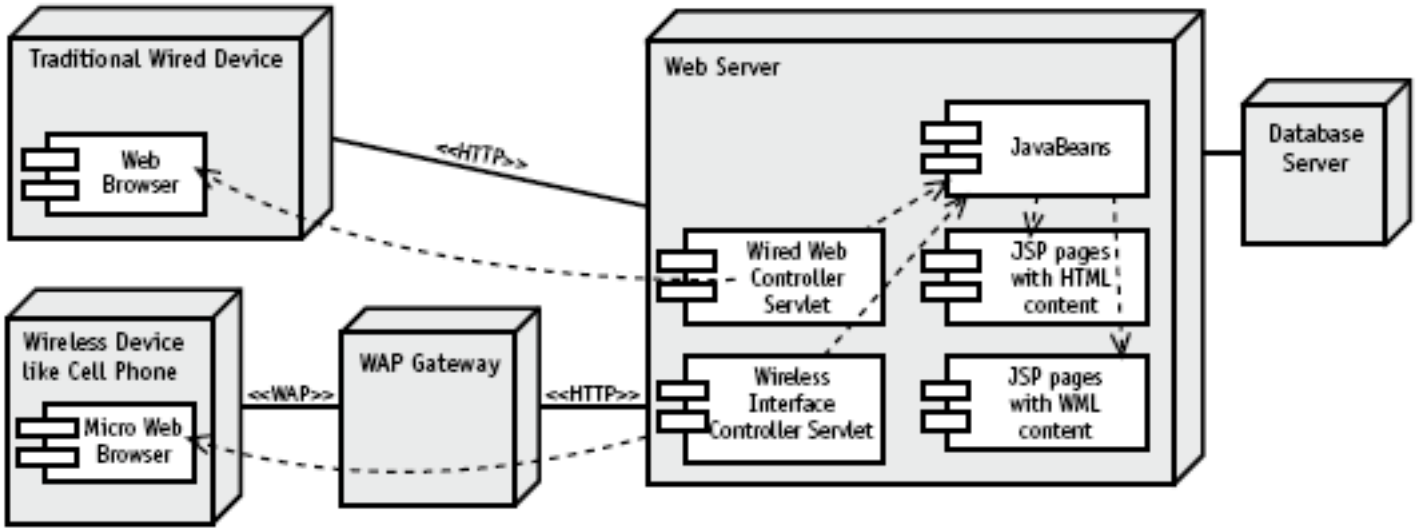
Ejemplo MVC: Vista Lógica de Arquitectura



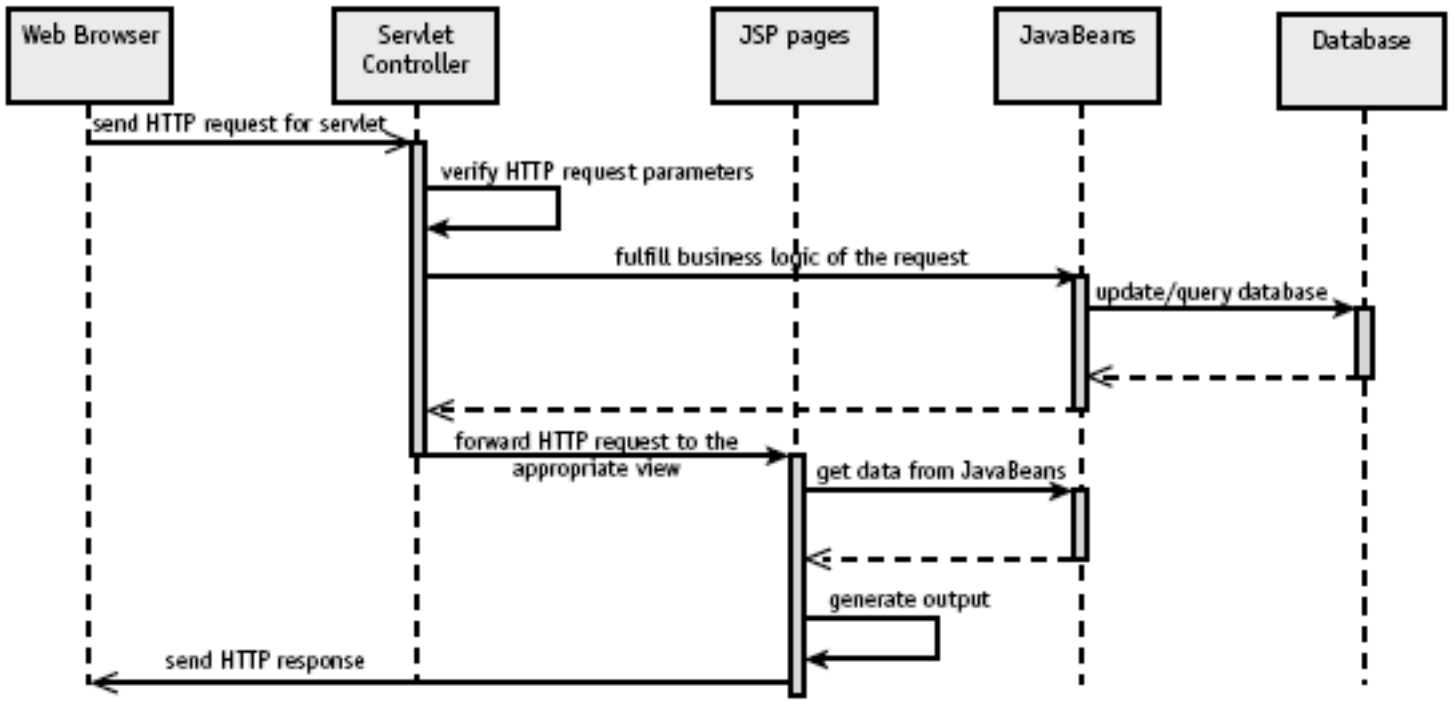
MVC típico en Java



Implementación MVC Java



Secuencia MVC en JAVA



Multiples clientes MVC