InforMAS
Revista de Ingeniería Informática del CIIRM
Dep.Legal: MU-2419-2004 ISSN: 1698-8841

Arquitectura y diseño de sistemas web modernos

por *Juan Salvador Castejón Garrido.*
Secretario del CIIRM.

Las aplicaciones web se han convertido en pocos años en complejos sistemas con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, dando servicio a procesos de negocio de considerable envergadura y estableciéndose sobre ellas requisitos estrictos de accesibilidad y respuesta. Esto ha exigido reflexiones sobre la mejor arquitectura y las técnicas de diseño más adecuadas. En este artículo se pretende dar un breve repaso a la arquitectura de tales aplicaciones y a los patrones de diseño más aplicables.

En los últimos años, la rápida expansión de Internet y del uso de intranets corporativas ha supuesto una transformación en las necesidades de información de las organizaciones. En particular esto afecta a la necesidad de que:

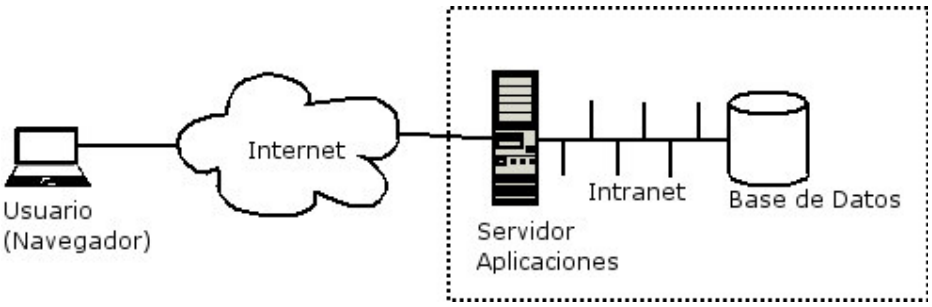
1. La información sea accesible desde cualquier lugar dentro de la organización e incluso desde el exterior.

2. Esta información sea compartida entre todas las partes interesadas, de manera que todas tengan acceso a la información completa (o a aquella parte que les corresponda según su función) en cada momento.

Estas necesidades han provocado un movimiento creciente de cambio de las aplicaciones tradicionales de escritorio hacia las aplicaciones web, que por su idiosincrasia, cumplen a la perfección con las necesidades mencionadas anteriormente. Por tanto, los sitios web tradicionales que se limitaban ha mostrar información se han convertido en aplicaciones capaces de una interacción más o menos sofisticada con el usuario. Inevitablemente, esto ha provocado un aumento progresivo de la complejidad de estos sistemas y, por ende, la necesidad de buscar opciones de diseño nuevas que permitan dar con la arquitectura óptima que facilite la construcción de los mismos.

El usuario interacciona con las aplicaciones web a través del navegador. Como consecuencia de la actividad del usuario, se envían peticiones al servidor, donde se aloja la aplicación y que normalmente hace uso de una base de datos que almacena toda la información relacionada con la misma. El servidor procesa la petición y devuelve la respuesta al navegador que la presenta al usuario. Por tanto, el sistema se distribuye en tres componentes: el navegador, que presenta la interfaz al usuario; la aplicación, que se encarga de realizar las operaciones necesarias según las acciones llevadas a cabo por éste y la base de datos, donde la información relacionada con la aplicación se hace persistente. Esta distribución se conoce como el modelo o arquitectura de tres capas.

Dep.Legal: MU-2419-2004 ISSN: 1698-8841



En la mayoría de los casos, el navegador suele ser un mero presentador de información (modelo de cliente delgado), y no lleva a cabo ningún procesamiento relacionado con la lógica de negocio. No obstante, con la utilización de applets, código javascript y DHTML la mayoría de los sistemas se sitúan en un punto intermedio entre un modelo de cliente delgado y un modelo de cliente grueso (donde el cliente realiza el procesamiento de la información y el servidor sólo es responsable de la administración de datos). No obstante el procesamiento realizado en el cliente suele estar relacionado con aspectos de la interfaz (como ocultar o mostrar secciones de la página en función de determinados eventos) y nunca con la lógica de negocio.

En todos los sistemas de este tipo y ortogonalmente a cada una de las capas de despliegue comentadas, podemos dividir la aplicación en tres áreas o niveles:

1. **Nivel de presentación**: es el encargado de generar la interfaz de usuario en función de las acciones llevadas a cabo por el mismo.

2. **Nivel de negocio**: contiene toda la lógica que modela los procesos de negocio y es donde se realiza todo el procesamiento necesario para atender a las peticiones del usuario.

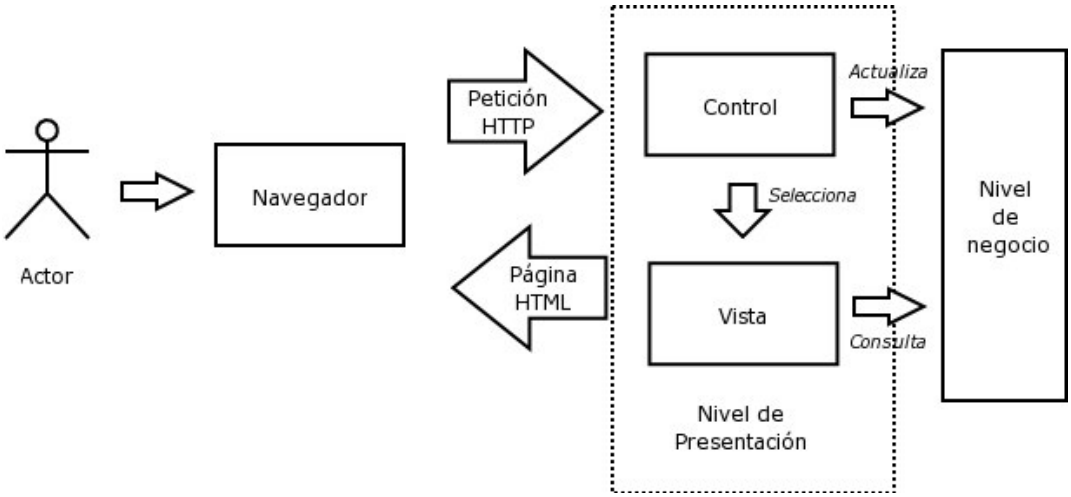
3. **Nivel de administración de datos**: encargado de hacer persistente toda la información, suministra y almacena información para el nivel de negocio.

Los dos primeros y una parte del tercero (el código encargado de las actualizaciones y consultas), suelen estar en el servidor mientras que la parte restante del tercer nivel se sitúa en la base de datos (notar que, debido al uso de procedimientos almacenados en la base de datos, una parte del segundo nivel también puede encontrarse en la misma). Teniendo en cuenta estas características en la arquitectura de los sistemas web, a continuación veremos algunos patrones de diseño de aplicación básica que pueden facilitar un diseño apropiado para este tipo de sistemas.

Uno de los patrones que ha demostrado ser fundamental a la hora de diseñar aplicaciones web es el **Modelo-Vista-Control (MVC)**. Este patrón propone la separación en distintos componentes de la interfaz de usuario (vistas), el modelo de negocio y la lógica de control. Una vista es una "fotografía" del modelo (o una parte del mismo) en un determinado momento. Un control recibe

un evento disparado por el usuario a través de la interfaz, accede al modelo de manera adecuada a la acción realizada, y presenta en una nueva vista el resultado de dicha acción. Por su parte, el modelo consiste en el conjunto de objetos que modelan los procesos de negocio que se realizan a través del sistema.

En una aplicación web, las vistas serían las páginas HTML que el usuario visualiza en el navegador. A través de estas páginas el usuario interactúa con la aplicación, enviando eventos al servidor a través de peticiones HTTP. En el servidor se encuentra el código de control para estos eventos, que en función del evento concreto actúa sobre el modelo convenientemente. Los resultados de la acción se devuelven al usuario en forma de página HTML mediante la respuesta HTTP.



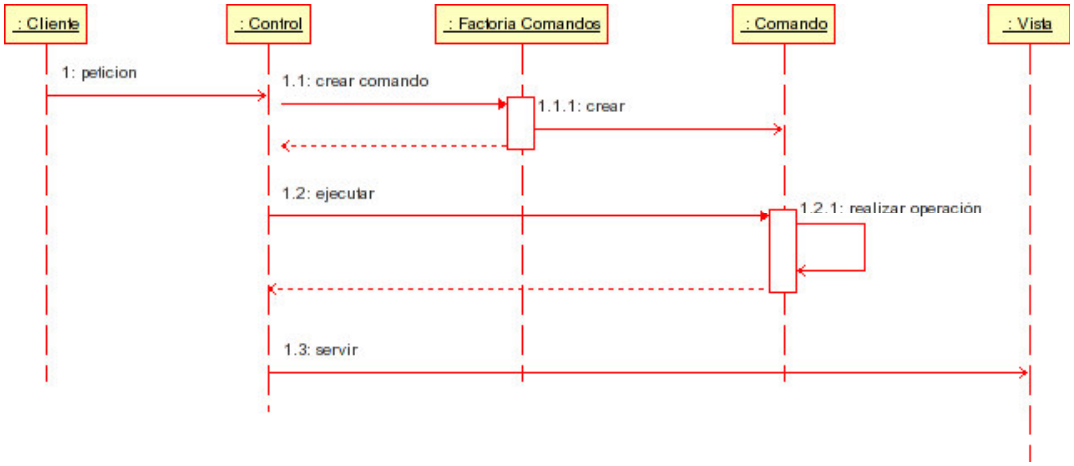
La clave está en la separación entre vista y modelo. El modelo suele ser más estable a lo largo del tiempo y menos sujeto a variaciones mientras que las vistas puede cambiar con frecuencia, ya sea por cambio del medio de presentación (por ejemplo HTML a WAP o a PDF) o por necesidades de usabilidad de la interfaz o simple renovación de la estética de la aplicación. Con esta clara separación las vistas pueden cambiar sin afectar al modelo y viceversa. Los controladores son los encargados de hacer de puente entre ambos, determinando el flujo de salida de la aplicación (qué se ve en cada momento).

Si tomamos como referencia la plataforma J2EE, las vistas podrían ser JSPs (o plantillas Velocity o documentos XML tratados con XSLT, ...) los controladores serían servlets y el modelo podría implementarse utilizando EJBs u objetos Java normales en combinación con frameworks de persistencia como Hibernate o JDO.

A la hora de utilizar el MVC en aplicaciones web es conveniente utilizar un único servlet como controlador para toda la aplicación. Este control gestiona todas las peticiones, incluyendo invocaciones a servicios de seguridad, gestión de excepciones, selección de la siguiente vista, etc. Esto se conoce como el patrón **Front Controller** (controlador frontal o fachada). El poder centralizar en

un solo punto servicios como la gestión de conexiones a base de datos, comprobaciones de seguridad o gestión de errores favorecen que la aplicación sea mucho más robusta y aísla de todos estos aspectos al resto de componentes.

Si aplicamos esto junto con el patrón **Command** podemos seguir lo que se conoce como **estrategia Comando y Controlador**. En este caso el componente Control (entendido dentro del marco MVC y no como el servlet controlador de este caso concreto) está formado por el servlet que recibe las peticiones y por un conjunto de clases implementadas a través del patrón **Command** en las que delega las tareas a llevar a cabo según la acción invocada. Estos comandos seleccionan la siguiente vista en función de los resultados de su procesamiento y el servlet controlador sirve esta vista al cliente.



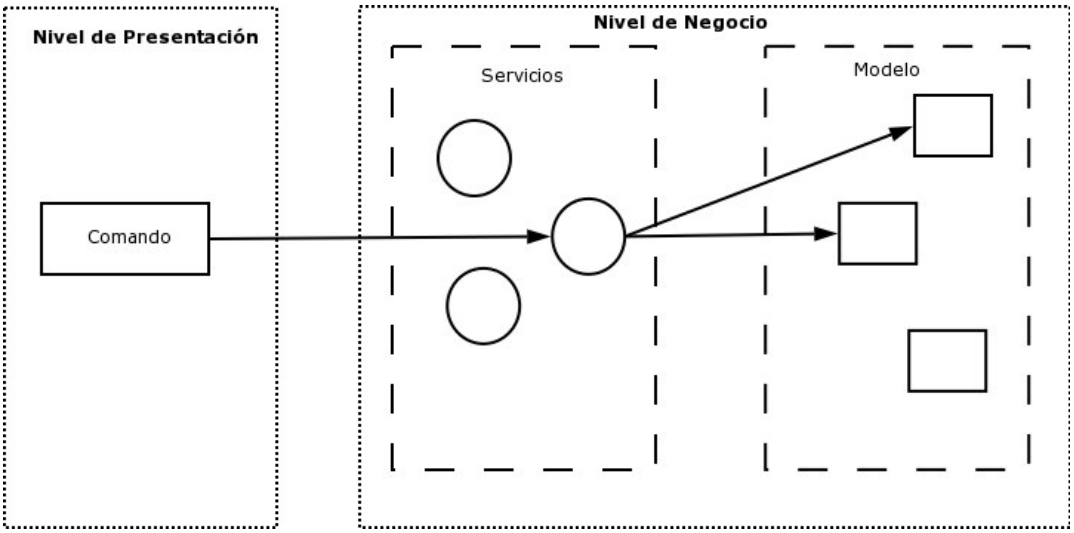
La utilización de esta estrategia tiene varias ventajas: permite cambiar fácilmente el procesamiento para una acción determinada sustituyendo el comando que la implementa por otro, permite reutilizar comandos y favorece el encadenamiento de dos o más comandos para la implementación de tareas complejas.

Actualmente existen varios frameworks de desarrollo de aplicaciones web basados en el patrón MVC como son Struts, Spring, WebWork o Maverick entre otros. El de mayor difusión en la actualidad es Struts. Este framework (al igual que muchos otros) utiliza la estrategia que hemos visto.

Si recordamos los tres niveles en los que dividimos una aplicación web (presentación, negocio, administración de datos), lo que hemos dicho hasta ahora afecta fundamentalmente al nivel de presentación y, en algunos casos, al nivel de negocio. Respecto a este último no hay ninguna “receta” para asegurar un buen diseño. La aplicación de patrones de diseño conocidos y el respeto a los principios de encapsulación de información y distribución de responsabilidad (que cada objeto haga solo aquello que le es propio) es la mejor manera de conseguir un diseño apropiado.

Dep.Legal: MU-2419-2004 ISSN: 1698-8841

Un principio que suele resultar de bastante utilidad es agrupar en servicios las operaciones de negocio relacionadas. Por ejemplo, en una aplicación de comercio virtual, podríamos tener un servicio para la gestión de usuarios, otro para la tramitación de pedidos, etc. De esta manera, si aplicamos la estrategia vista anteriormente, los comandos invocarían uno o más métodos de estos servicios, que serían los que accederían a los objetos que constituyen el modelo.



Por último hablaremos brevemente del nivel de administración de datos. Este último nivel es proporcionado por el framework de persistencia utilizado junto con la base de datos propiamente dicha. Actualmente existen diversos frameworks de persistencia de datos (Entity beans, Hibernate, JDO, OJB, etc.) que realizan automáticamente el mapeo entre objetos de la aplicación y tablas en la base de datos relacional o incluso podríamos optar por utilizar JDBC. Cada uno tiene unas características y funcionalidad concretas que obligarán a adaptar el diseño de manera apropiada. Solo apuntar que normalmente, en una aplicación web una petición HTTP equivale a una transacción. Es decir, si mientras se sirve la petición ocurre un error todo lo hecho a raíz de esa petición debe deshacerse. Por tanto, en función del modelo de persistencia, habrá que actuar de manera que los cambios a la base de datos no se hagan definitivos hasta que la petición se haya completado. La mejor forma de gestionar esto es en el servlet controlador, pues al ser el que finaliza la petición enviando la respuesta al cliente, es el mejor lugar para estar seguros de que todo ha ido bien y hacer definitivos los cambios en la base de datos.

Con esto completamos el breve análisis de la arquitectura de los sistemas web modernos, donde hemos dado apuntes para un diseño lo más correcto posible de los mismos, sobre todo en lo referente al nivel o capa de presentación.

Sólo me queda pues despedirme deseando lo mejor a esta nueva revista que comienza su andadura y felicitando a todos aquellos que la han hecho posible.

Fecha: 12/01/2004

InforMAS
Revista de Ingeniería Informática del CIIRM
Dep.Legal: MU-2419-2004 ISSN: 1698-8841

Recursos relacionados:

- java.sun.com: Java 2 Platform, Micro Edition (J2ME) http://java.sun.com/reference/blueprints/
- J2ME Connected Limited Device Configuration (CLDC); JSR 30, JSR 139 http://www.uidesign.net/Articles/Papers/UsingMVCPatterninWebInter.html
- E. Gamma et al, Design Patterns, Elements of reusable software. Addison Wesley Professional. 1995
- Ian Sommerville, Ingeniería de Software. Sexta Edición. Addison Wesley. 2002