Definición de un Framework para aplicaciones Web con navegación sensible a concerns

Ing. Iván Mendoza Vázquez

Trabajo de tesis para la obtención del título

De Máster en Ingeniería de Software

Facultad de Informática

Universidad Nacional de La Plata

República Argentina

Agradecimientos

A mi esposa Elina y su familia por el apoyo brindado durante este tiempo

A mis hijos Julián Andrés y Ezequiel, que es por ellos que sigo adelante

A mis padres y hermanos

A mi director de tesis, por no perder la fe en que la terminaría.

Contenido

1. Introducción y Objetivos

1.1 Introducción

El estado actual de las aplicaciones Web existentes, es el de servicios enfocados a ofrecer mejores soluciones al usuario final, brindando un ambiente similar al de las aplicaciones comunes de escritorio. Para esto hoy en día es casi obligado el uso de técnicas como: hojas de estilo, un marcado XHTML válido semánticamente, menor refrescamiento de pantalla mediante AJAX, mayor intervención de programación del lado del cliente con Javascript, URLs limpias con significado semántico, entre otras; que resultan en aplicaciones más dinámicas e intuitivas para el usuario.

Todo esto en contraste con el estado de la Web inicial, en la que el usuario se encontraba en un entorno estático, con páginas HTML (sin programación del lado del servidor) que sufrían pocas actualizaciones y no tenían interacción con el usuario.

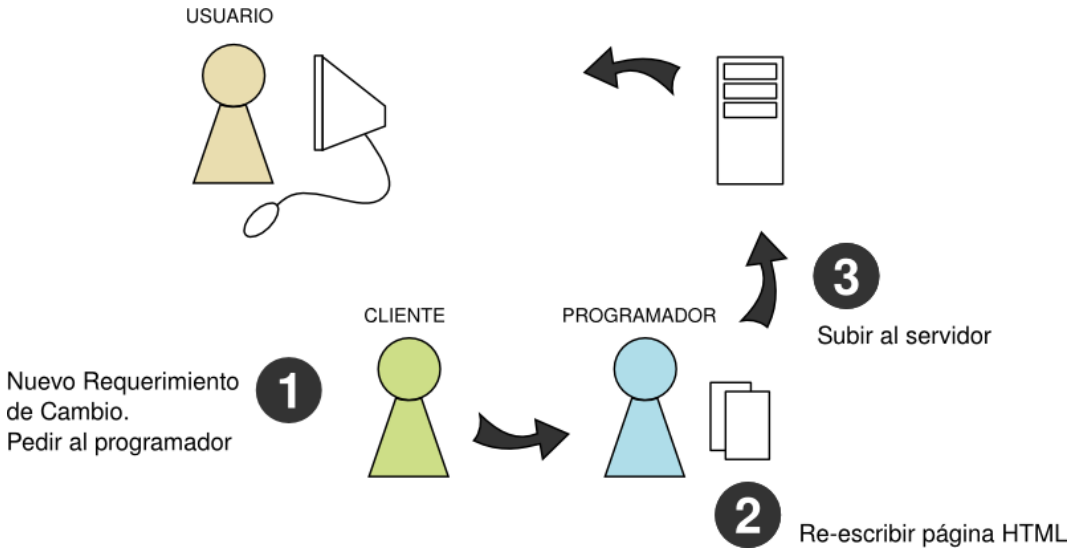


Ilustración 1. Forma tradicional de actualización de contenidos

Hace algunos años la construcción de sitios Web era exclusiva para desarrolladores con conocimientos de programación o de diseño, y el producto obtenido no siempre

satisfacía las necesidades del cliente o del usuario final. Esto cambió radicalmente al entrar a escena los gestores de contenidos, que permiten la edificación rápida de un sitio Web, así como la actualización de sus contenidos de manera dinámica obteniendo la información desde una o varias bases de datos.

Ilustración 2. Actualización de contenidos a través de un CMS

La principal ventaja de estos sistemas, es que es un usuario la persona que realiza todo el desarrollo sin necesidad de poseer conocimientos técnicos, y de una manera relativamente sencilla. En los últimos tiempos estos sistemas han evolucionado hacia wikis y redes sociales, donde los contenidos se actualizan de forma colaborativa por varios usuarios dentro de una comunidad y cuyo éxito han transformado la manera en que se usaba Internet.

Además de las técnicas de programación antes mencionadas, existen otras que no son nuevas en este campo pero que su uso estaba limitado al diseño de la arquitectura de las aplicaciones; por lo que sus resultados no eran visibles para el usuario final.

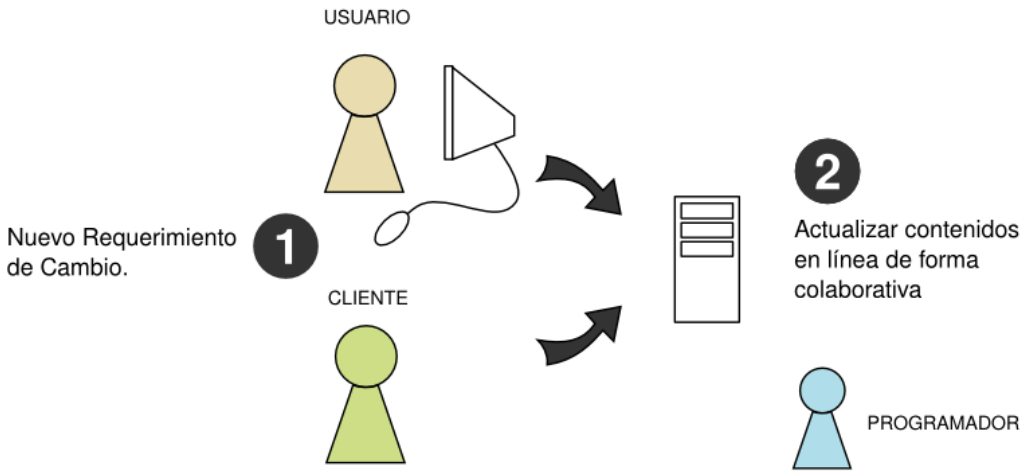


Ilustración 3. Actualización de contenidos a través de redes sociales

Una de estas técnicas es la separación de concerns, que aunque inicialmente fue concebida para permitir la evolución y escalabilidad del software, bien puede servir para enriquecer la experiencia del usuario final y para resolver ciertos problemas de navegación que afectan todavía a la mayoría de los sitios en Internet.

La forma en que esta técnica colabora a alcanzar esta meta, es personalizando los enlaces y contenidos en base a los intereses del usuario para conseguir una navegación más lógica e intuitiva, consiguiendo que cada usuario explore el sitio de forma diferente. Este aspecto será denominado a lo largo de este documento como Navegación sensible a concerns o CSN.

En los próximos capítulos discutiremos su diseño, implementación y los resultados obtenidos durante su integración con las aplicaciones Web más comunes en la actualidad, todo esto con el fin de lograr la construcción rápida y ordenada de las mismas.

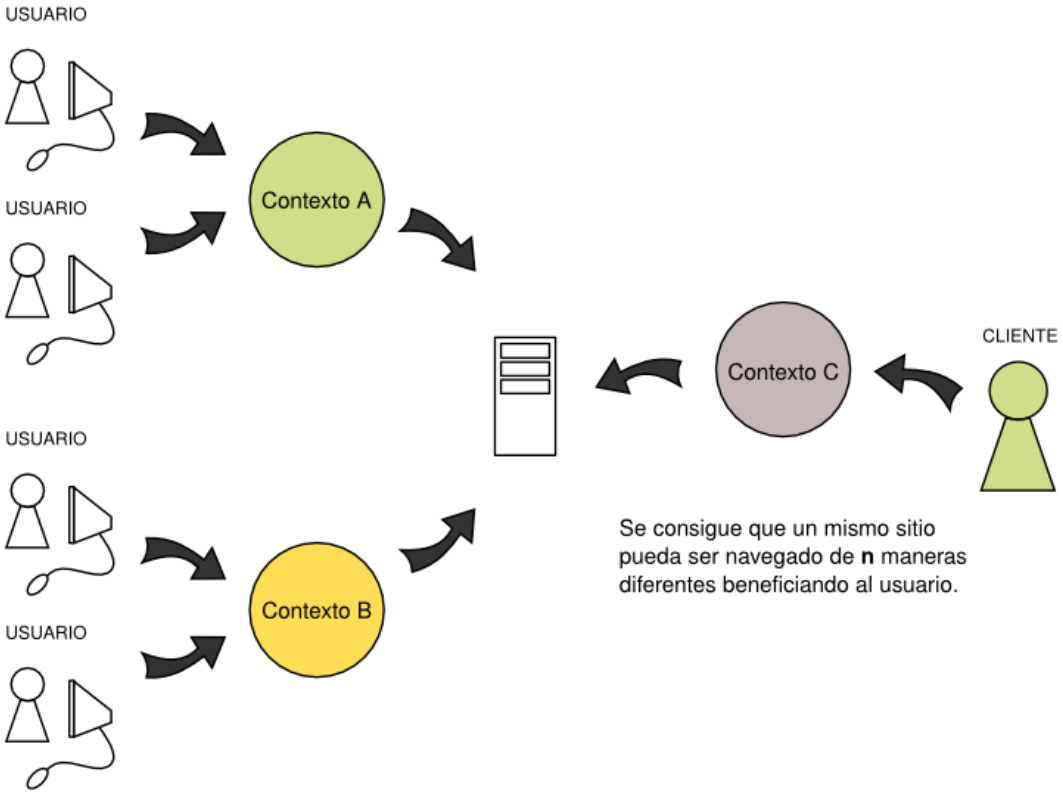


Ilustración 4. Navegación Sensible a Concerns

1.2 Objetivo

El propósito de esta tesis es demostrar que es posible crear una estructura de soporte definida, que de ahora en adelante llamaremos Framework, cuyo núcleo es un conjunto de clases o librerías mediante las cuales se pueda desarrollar aplicaciones Web con navegación sensible a concerns de manera rápida, ordenada y segura.

1.3 Contribuciones

Partiendo del objetivo fundamental explicado en el punto anterior, obtenemos las siguientes contribuciones:

- Dar a conocer el estado actual de los sitios y aplicaciones Web existentes, y sus principales problemas de navegación.

- Brindar alternativas basadas en separación de concerns a los esquemas de navegación convencionales, para mejorar la experiencia del usuario final.
- Mostrar las ventajas y desventajas de adoptar CSN
- Proponer una estructura definida para el desarrollo de aplicaciones web con CSN
- Demostrar la vialidad de migrar una aplicación existente a un esquema CSN
- Demostrar la utilización del Framework obtenido mediante la construcción de un prototipo
- Proponer nuevos trabajos en esta misma línea para mejorar el estado de la Web actual

1.4 Alcance

Este trabajo involucra tanto el diseño como la implementación de un Framework orientado a objetos, así como la demostración y factibilidad de su utilización mediante la construcción de una aplicación prototipo para la gestión de contenidos de un sitio Web informativo.

Si bien el Framework será edificado desde el inicio, la arquitectura modular que se busca en su diseño permite que el soporte para CSN pueda ser acoplado en otros frameworks MVC comerciales.

El alcance se resume en los siguientes puntos:

- Análisis de los problemas de navegación actual junto con sus posibles soluciones al emplear sensibilidad a concerns.
- Desarrollar un Framework para aplicaciones Web con soporte CSN integrado.
- Diseñar una aplicación de gestión de contenidos (CMS) enriquecida con CSN
- Construir el CMS usando como base el Framework desarrollado.
- Revisar los resultados obtenidos y establecer trabajos futuros para ésta y otras aplicaciones Web.

1.5 Organización del Texto

Este documento consta de 10 capítulos explicados a continuación:

Capítulo 1.- Una presentación hacia lo que va a tratar la tesis, junto con los objetivos que se desean alcanzar.

Capítulo 2.-Un análisis de los problemas de navegación en los sitios y aplicaciones Web actuales, junto con las alternativas de mejora que brinda el esquema de navegación sensible a concerns.

Capítulo 3.- Buenas prácticas para la definición de estructuras de navegación en algunas de las aplicaciones Web 2.0 de la actualidad.

Capítulo 4.- Fundamentos de la separación de concerns en el desarrollo de aplicaciones, explicación de la navegación sensible a concerns como herramienta para mejorar la experiencia del usuario y los trabajos relacionados en este tema.

Capítulo 5.- Enriquecimiento de las aplicaciones Web 2.0 mediante navegación sensible a concerns.

Capítulo 6.- Diseño de un Framework con soporte para CSN, sus principales componentes y documentación sobre la utilización del mismo.

Capítulo 7.-Diseño y resultados de la implementación de un gestor de contenidos, como prototipo para demostración del Framework concebido en este proyecto.

Capítulo 8.- Trabajos futuros y propuestas a partir de los resultados obtenidos conclusiones generales.

Capítulo 9.- Conclusiones generales.

Capítulo 10.- Bibliografía.

2. Problemas de Navegación

2.1 Origen del Problema

"A menos que un sitio Web tome en cuenta las necesidades de sus usuarios finales, éste no cumplirá con los requerimientos de la organización que provee el sitio". Esta frase nos sugiere que el desarrollo de sitios Web debería estar centrado en el usuario, contemplando un diseño evolutivo que avance basado en sus requerimientos.

El primer paso para lograr este propósito es definir los objetivos del negocio, es decir, los posibles escenarios o contextos de uso. El diseño del sitio debería tomar en cuenta las líneas guía establecidas para la navegación, escritura y la forma de cada página. Actualmente existen millones de sitio Web que carecen de los requisitos mínimos para brindar una experiencia satisfactoria a sus usuarios, no solo desde el punto de vista gráfico, sino que son difíciles de mantener y actualizar. Las razones de esto pueden incluir lo siguiente:

Las organizaciones frecuentemente producen sitios con contenido y estructura que reflejan aspectos internos de la organización en lugar de las necesidades reales de sus lectores.

Los sitios Web normalmente contienen material que sería apropiado para un formulario impreso, pero que necesita ser adaptado para la presentación en la Web. Resulta muy difícil para los usuarios encontrar lo que buscan de una manera rápida y sencilla al no tener información importante a la mano, o al no tener enlaces que lleven a esa información cuando se requiera.

Producir páginas Web es aparentemente una tarea fácil, por lo que las personas que las desarrollan no tiene los mismos criterios de calidad que son usados para formas más tradicionales de publicidad. Cuando Internet apareció, la demanda de empresas que querían tener presencia en este mundo creció bruscamente, dando como resultado la construcción masiva de sitios Web sin dejar tiempo para el planteamiento de estándares y buenas prácticas.

No se consideran los posibles escenarios o contextos de uso por lo que toda página se presenta exactamente de la misma manera sin importar como se llegó a ella. Esto produce una deficiente estructura de navegación, ya que los enlaces y operaciones incluidas en ésta son siempre los mismos, sin tomar en cuenta lo que realmente el usuario está buscando. Si pudiéramos brindar al usuario diferentes esquemas de navegación, los caminos para llegar a la página buscada serían más cortos una vez que tenemos bien identificado el contexto en el que nos movemos.

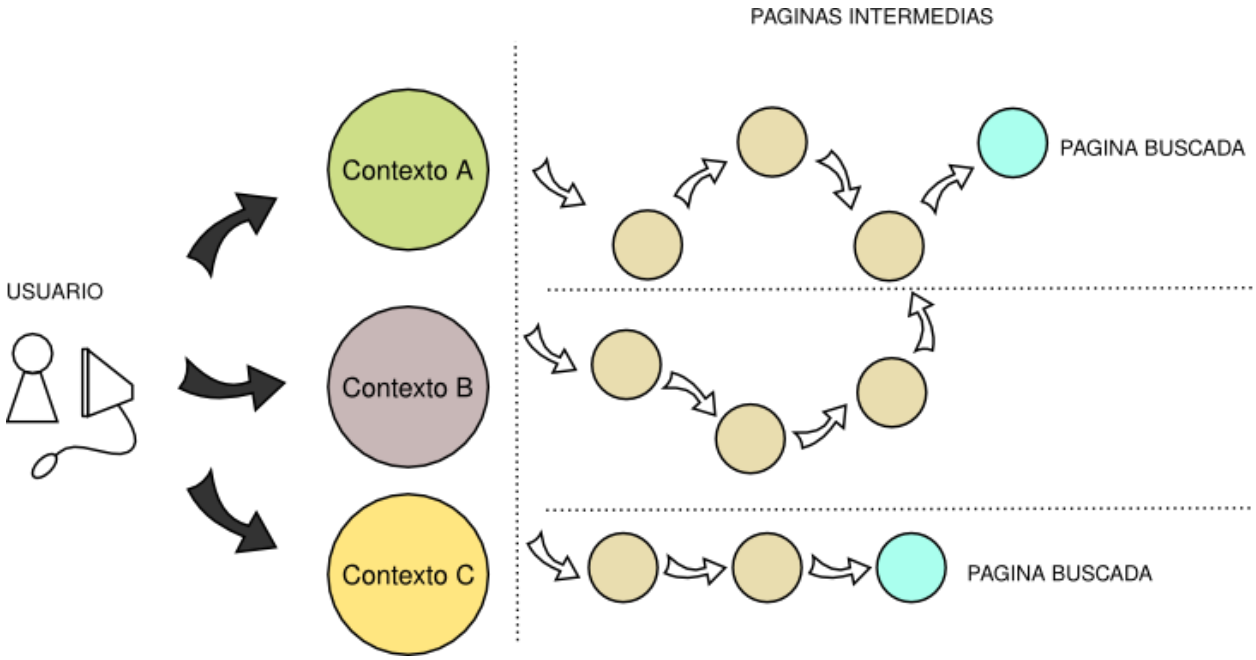


Ilustración 5. Navegación con diferentes esquemas

Vamos a tomar como ejemplo para éste análisis, sitios de comercio en línea (e-commerce) en los que el usuario ingresa para buscar un producto según sus necesidades y comprarlo. Estos sitios frecuentemente carecen de una estructura adecuada de navegación y son difíciles de usar lo que produce que el usuario se canse y opte por finalizar la búsqueda. Analizaremos los principales problemas y luego veremos la manera de mejorar la experiencia del usuario corrigiendo solo aquellos relacionados con la navegación.

2.2 Problemas recurrentes en sitios comerciales

A continuación se mencionan las equivocaciones más frecuentes en estos sitios,desde el punto de vista de la navegación, junto con la posible solución basada en CSN para evitarlas o corregirlas y de esta manera mejorar la experiencia del usuario final.

2.2.1 Falta de información detallada de un producto

Cuando se compra en un almacén de verdad, se tiene la ventaja de poder tomar un producto, sentirlo, verlo desde cada ángulo y leer cualquier información en las etiquetas o envolturas. En sitios e-commerce esta interacción se elimina, y se necesita hacer todo lo posible para mejorar la experiencia de compra.

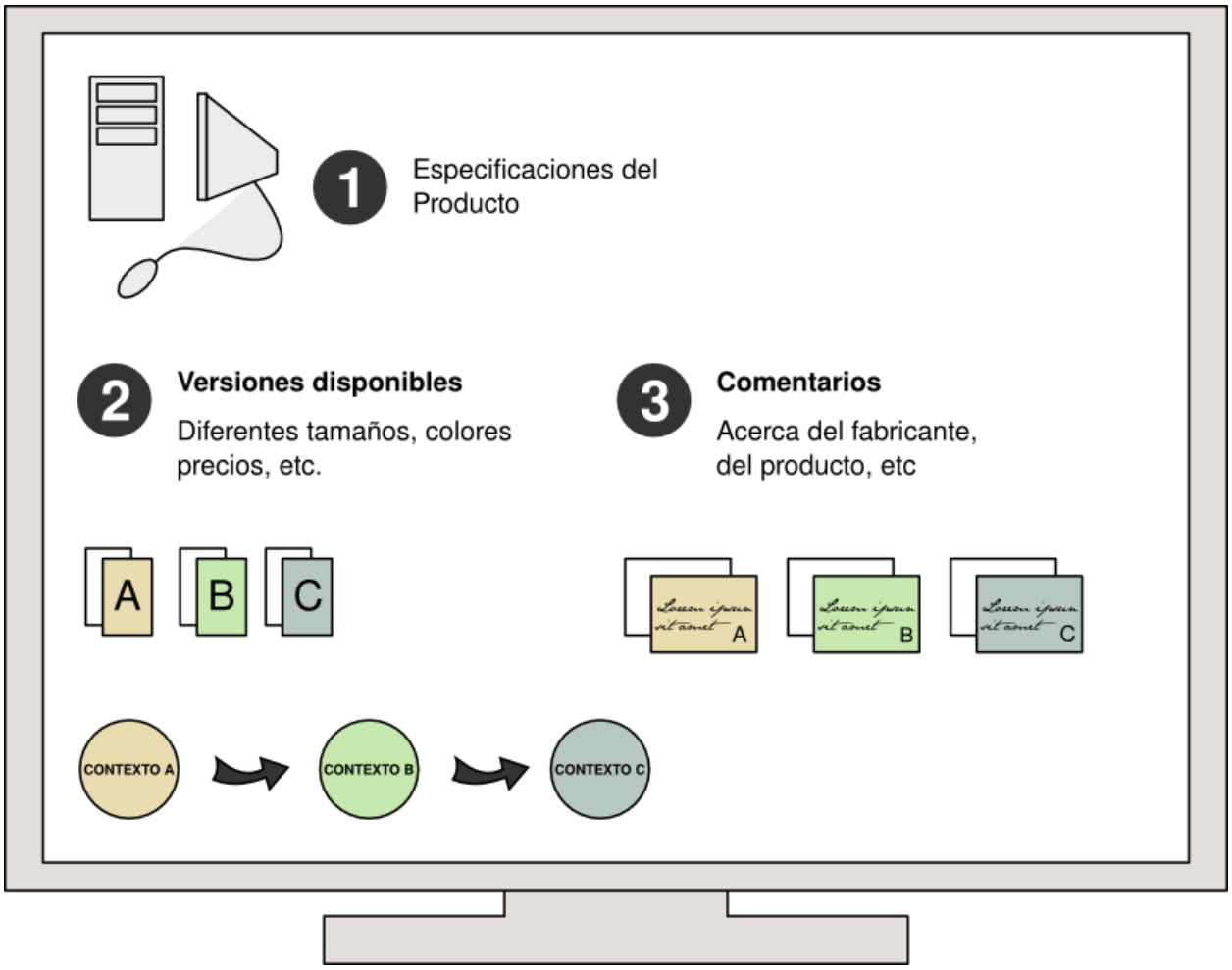


Ilustración 6. Falta de información detallada

Lo que se debe hacer es proveer tanta información como se pueda como: tamaños disponibles, materiales, peso, dimensiones; comentarios sobre el fabricante, el diseñador o experiencias de usuarios que previamente compraron el producto, etc.

Estas mejoras pueden ser incrementadas si en lugar de mostrar información en exceso, se la filtra en base a un contexto de navegación. De esta manera solo se muestra lo concerniente al contexto actual, pudiendo pasar a otro en cualquier momento.

Un ejemplo podría ser navegación por fabricante o marca, donde las versiones disponibles no solo incluirían al mismo dispositivo, sino a otros dispositivos similares del mismo fabricante, y los comentarios que aparecen únicamente hablarían de la reputación y buena atención (feedback) del mismo.

2.2.2 Ocultar información de contacto

Los consumidores desean saber que ellos están tratando con una compañía real cuando usan la información de su tarjeta de crédito. Ellos quieren saber que si llega a presentarse un problema podrán comunicarse con una persona real y conseguir la ayuda que necesitan. Si el sitio no provee ninguna información de contacto, o la oculta de manera que el usuario no la pueda encontrar fácilmente, es probable que no confíen en el sitio y por lo tanto que no hagan negocios en él.

Lo que se debe hacer es colocar nuestra información de contacto en un lugar fácil de encontrar en cada página del sitio. Los lugares más obvios para colocar ésta información pueden ser la cabecera, la parte superior de la barra lateral, o en el pie de página. Además proveer múltiples medios de contacto si es posible, un formulario de contacto, dirección de email, número telefónico y cualquier información que aumente el nivel de confianza del usuario.

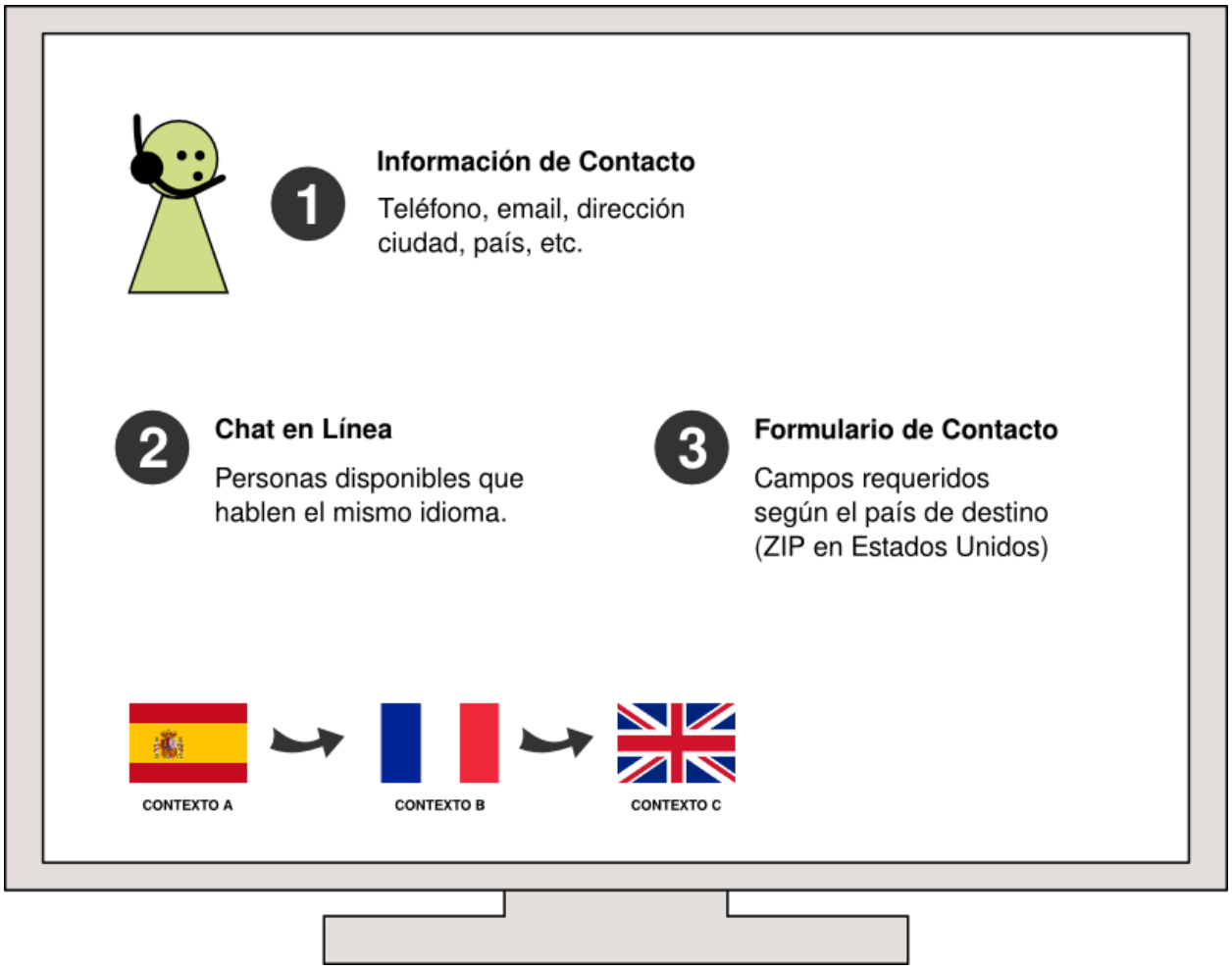


Ilustración 7. Ocultar información de contacto

Esta información de contacto podría ser sensible al contexto actual, en el caso de sitios donde la primera página obligue a la selección del país o región del usuario, un ejemplo de estos datos sería: dirección y teléfono de un representante en el mismo país, chat en línea en el mismo idioma del usuario, formulario con campos que solo apliquen a esa zona (Zip en el caso de Estados Unidos).

2.2.3 Requerir una cuenta para ordenar

Si se requiere que el cliente se registre y cree una cuenta antes que pueda realizar su orden, esto se convierte en otro obstáculo que hemos colocado en su camino. Deberíamos preguntarnos qué es más importante: obtener el pedido o capturar información del cliente, tomando en cuenta que la segunda implica perder algunos clientes potenciales.

Lo que se debe hacer es ofrecer esta opción al cliente al final del proceso de compra, en lugar de pedir que se registre antes de la orden. Se debe dar la opción de guardar su información indicando que es con el fin de realizar órdenes futuras más fácilmente o para poder realizar un seguimiento de su pedido actual, la mayoría optarán por aceptar y no habremos perdido clientes antes de haber completado el pedido.

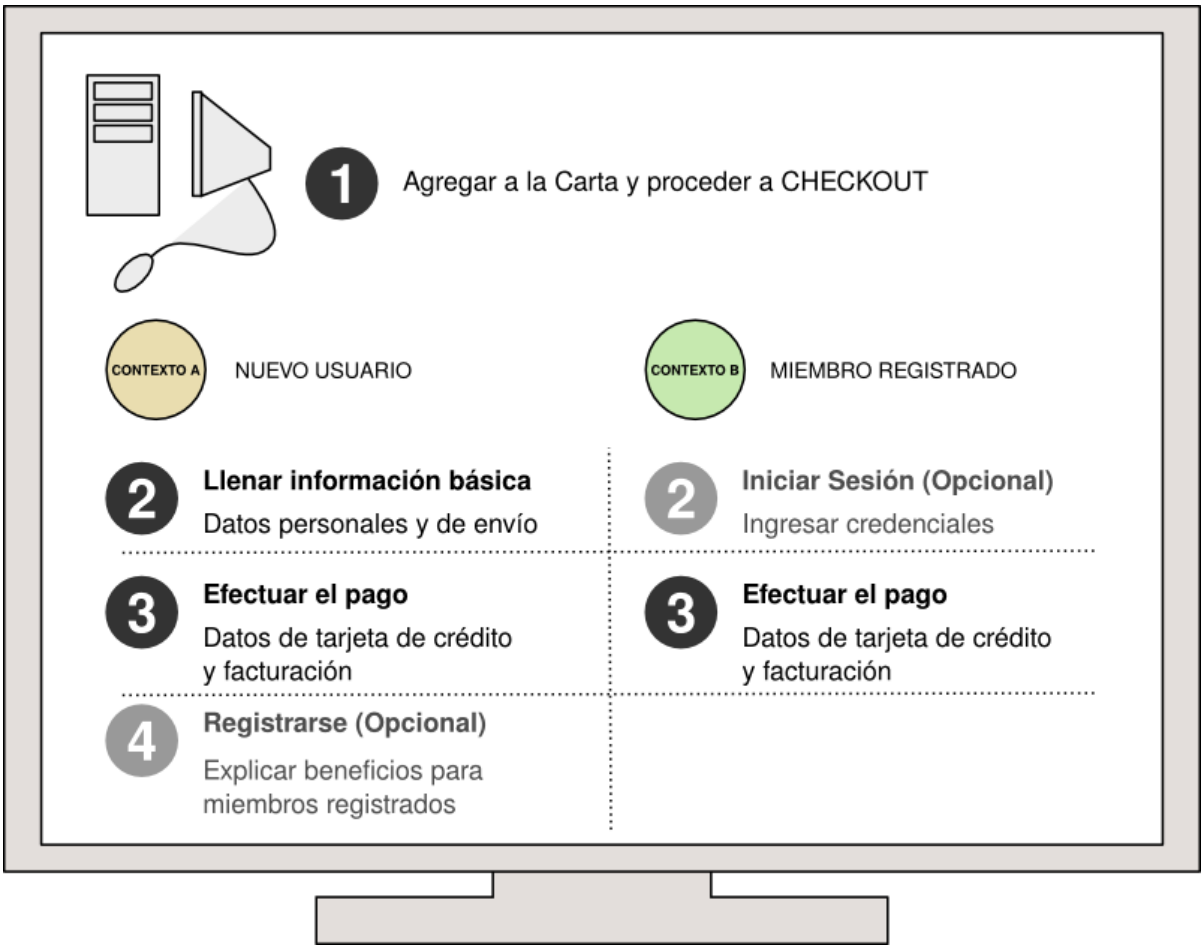


Ilustración 8. Requerir una cuenta para ordenar

En este punto podemos identificar 2 contextos principales, en el primero el usuario no es un miembro registrado en el sitio y en el segundo si lo es. En ambos casos el esquema de navegación es diferente y por lo tanto también las operaciones presentadas al usuario, pero al final el pedido toma lugar y el producto es comprado.

Algunos pasos han sido marcados como opcionales, puesto que no son obligatorios como el número 4 del primer contexto, o pueden ya haber ocurrido con anterioridad como el número 2 para miembros registrados. Además podrían presentarse más operaciones o incluso más esquemas de navegación en procesos más complejos.

2.2.4 Un inadecuado motor de búsqueda

Si un cliente sabe exactamente lo que está buscando, muchos optarán por usar un motor de búsqueda en lugar de navegar a través de categorías y menús. Por esta razón necesitamos asegurarnos de que el buscador en el sitio sea fácil de utilizar y que preferiblemente tenga filtros posteriores que permitan refinar los resultados.

Analicemos esta situación. Pensemos qué tan seguido hemos buscado un producto en un sitio e-commerce grande y hemos obtenido cientos de resultados. Aunque la variedad de opciones debería ser algo bueno, si la mitad de esos resultados no son lo que estamos buscando, esto viene a ser más bien un inconveniente. Sin embargo incluir filtros posteriores por algún criterio a los resultados ya obtenidosdisminuye el problema.

Lo que se debe hacer es incluir estos filtros posteriores a la búsqueda, y además permitir que los usuarios puedan ordenar sus resultados basados en criterios como: más populares, mayor o menor precio, productos nuevos, etc. Además se podría pensar en darle la opción de especificar cuántos resultados por página quiere ver.

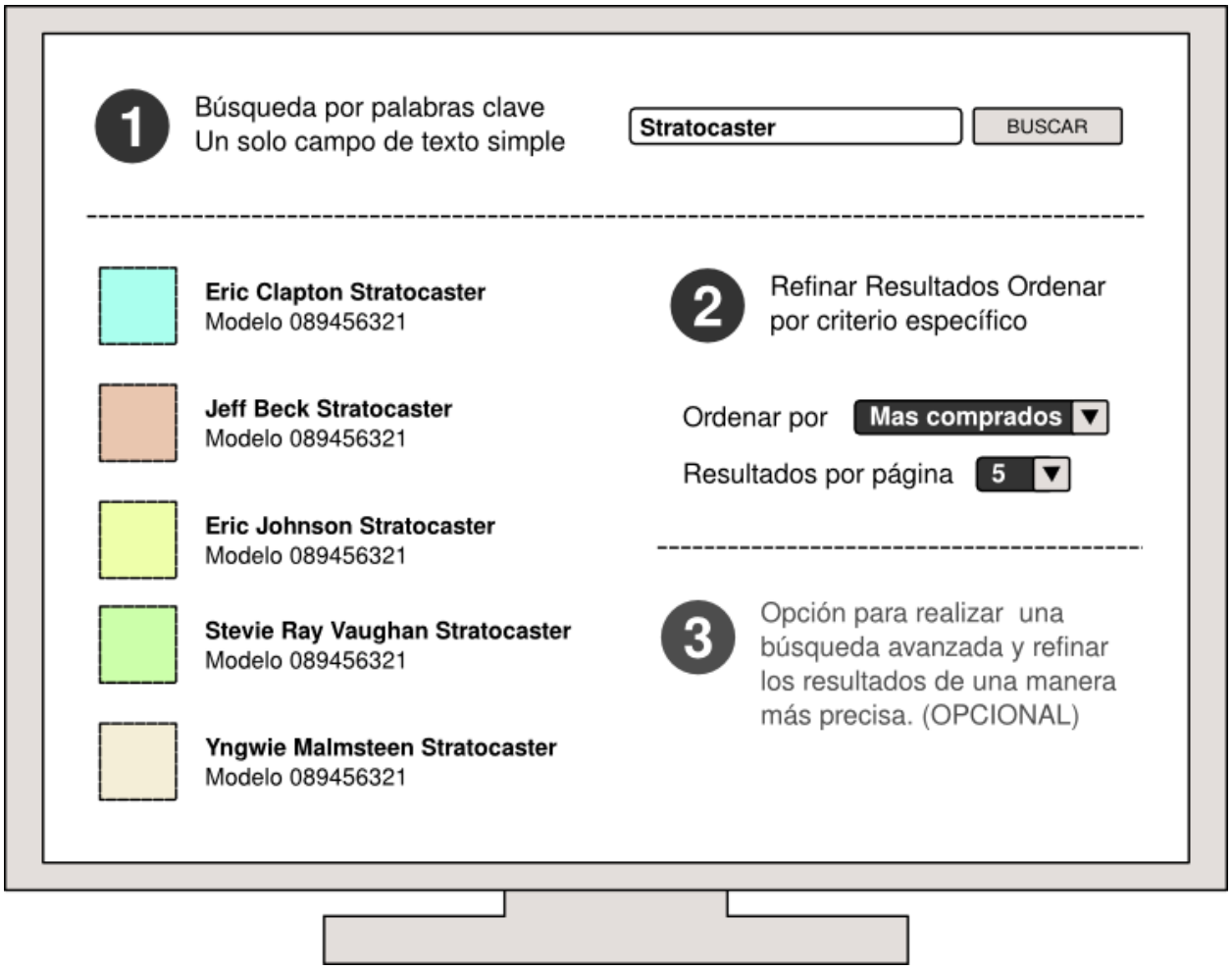


Ilustración 9. Motor de Búsqueda

Se puede enriquecer aún más la experiencia del usuario si modificamos la interfaz de usuario, para dejar que los resultados de la búsqueda estén presentes a un costado de la página mientras se sigue navegando; de ésta manera el usuario podrá seguir revisándolos sin tener que regresar a la página anterior u obligándolo a realizar nuevamente una búsqueda. Por lo tanto, dado un nuevo esquema de tipo búsqueda re-estructuramos la página para facilitar su navegación.

Además el ordenamiento por defecto de los resultados podría ser diferente para un contexto diferente.

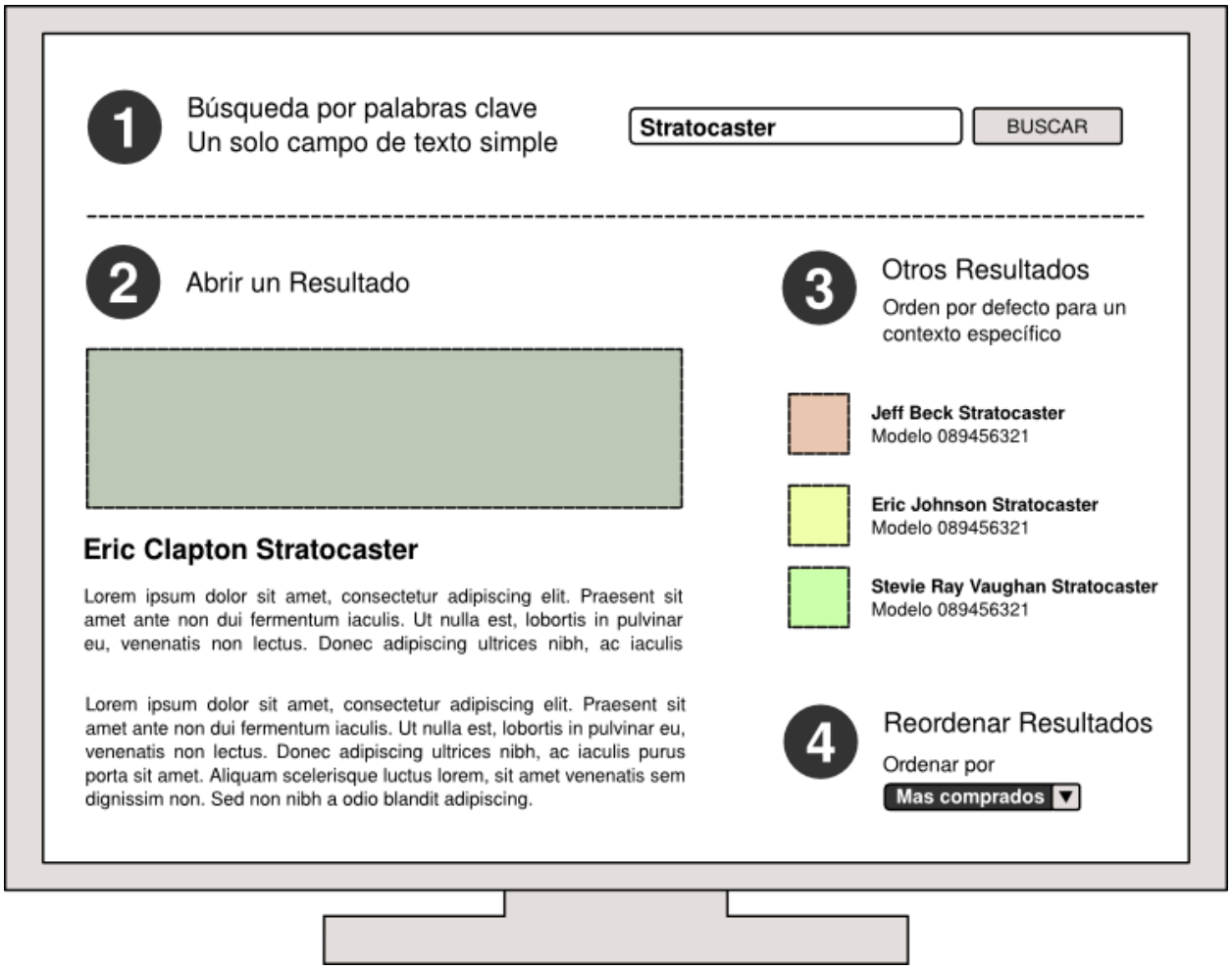


Ilustración 10. Motor de búsqueda mejorado

2.2.5 Un mal diseño de Carro de Compras

El carro de compras (shopping cart) aparte del servicio de búsqueda es el componente más importante de un sitio de comercio en línea, de él depende el éxito o no del sitio. Entre sus principales funciones, se necesita permitir a los usuarios ordenar múltiples productos rápidamente, revisar sus cantidades o eliminarlos de la orden. Además que debe ser fácil de usar y tanto su contenido como el total a pagar deben estar visibles todo el tiempo.

Lo primero que debemos asegurar es que el carro de compras deje al usuario agregar un producto y luego regresar con facilidad a la última página donde estaba para continuar con su compra; o aún mejor, permitirle agregar un producto al carro sin incluso dejar la página donde está, por medio del uso de un mini carro situado en una parte visible de la pantalla. Adicionalmente podemos permitir a los clientes editar las cantidades de los productos o removerlos del carro todo desde el mismo lugar.

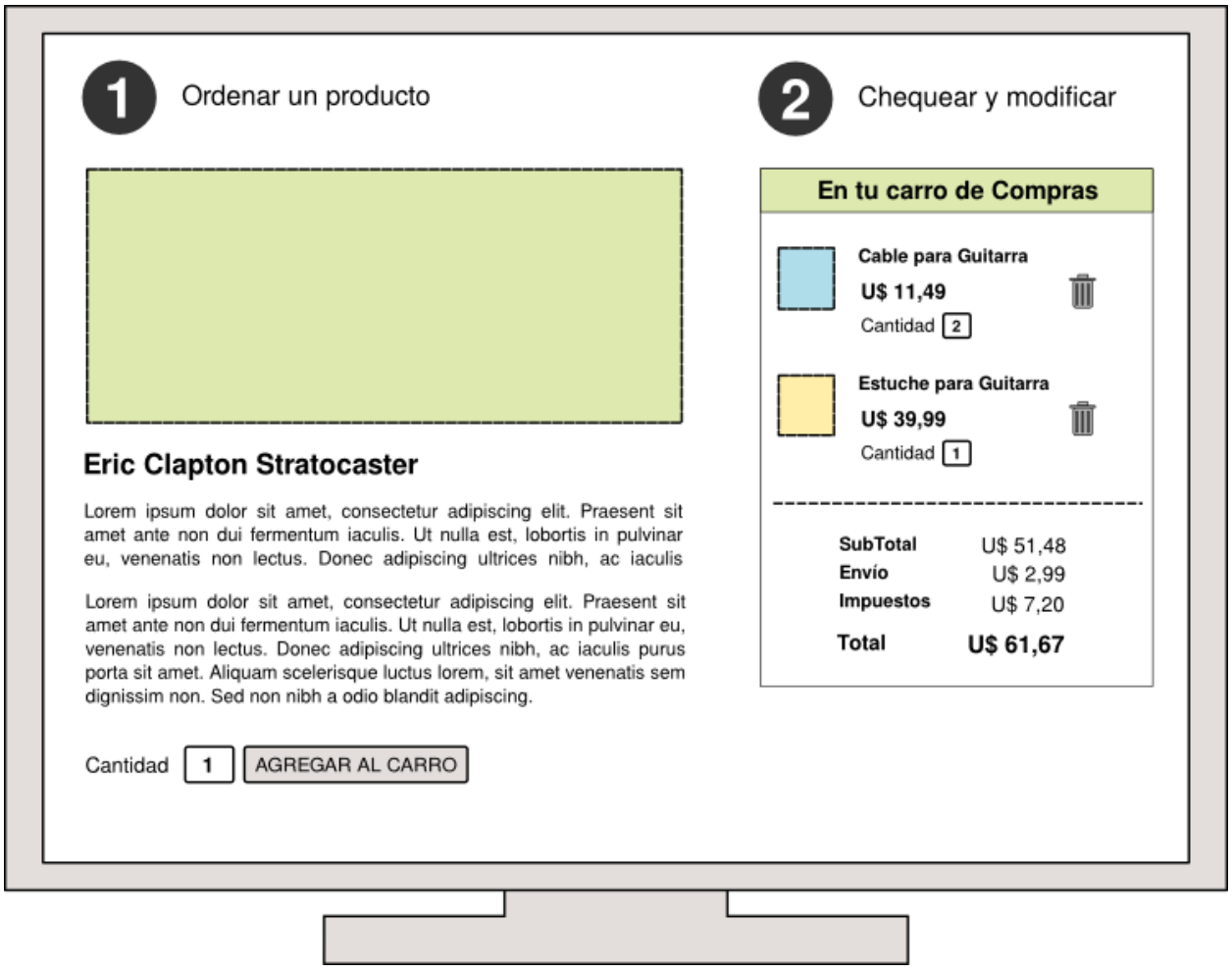


Ilustración 11. Diseño aceptable para un carro de compras

Finalmente se debería tener a la vista los cargos de envío y totales antes de incluso empezar el proceso de Checkout. Todos los montos deberán ser automáticamente actualizados cuando se modifique una cantidad o ítem. La posición del mini carro puede variar en la página dependiendo del esquema actual de navegación, lo importante es que esté siempre presente y que sea fácil de encontrar por el usuario.

2.2.6 No Incluir Productos Relacionados

Probablemente hayamos notado que cuando vamos a un almacén cualquiera, los productos que se complementan están colocados siempre juntos unos a otros, haciendo más fácil encontrar combinaciones e incentivando al cliente a llevar más de lo que realmente estaba buscando (por ejemplo en el almacén será común encontrar los estuches de celulares cerca de los teléfonos celulares, o toda la ropa de hombre agrupada en un mismo sector). Lo mismo se puede hacer en un sitio Web y esto disminuye el tiempo de navegación del usuario y puede aumentar las ventas para nuestro negocio.

No estamos refiriéndonos a simplemente agrupar artículos por categorías y subcategorías, sino a colocar a la vista del usuario productos afines al momento que revisar uno en particular. Así mismo no solo se trata de accesorios para el producto, sino de combinaciones con otros artículos.

Esta práctica no está limitada solo para sitios de ventas en línea, la misma idea se puede plantear para otros casos como:

- Diarios en línea, que muestran noticias relacionadas a la revisada en ese momento
- Blogs personales que muestran los postings relacionados con el revisado, por fecha, por tema, o sugeridos por el autor.
- Buscadores de empleo, que muestran los empleos relacionados con el revisado por el usuario, bien sea por similar destreza o que se encuentran en la misma ciudad.

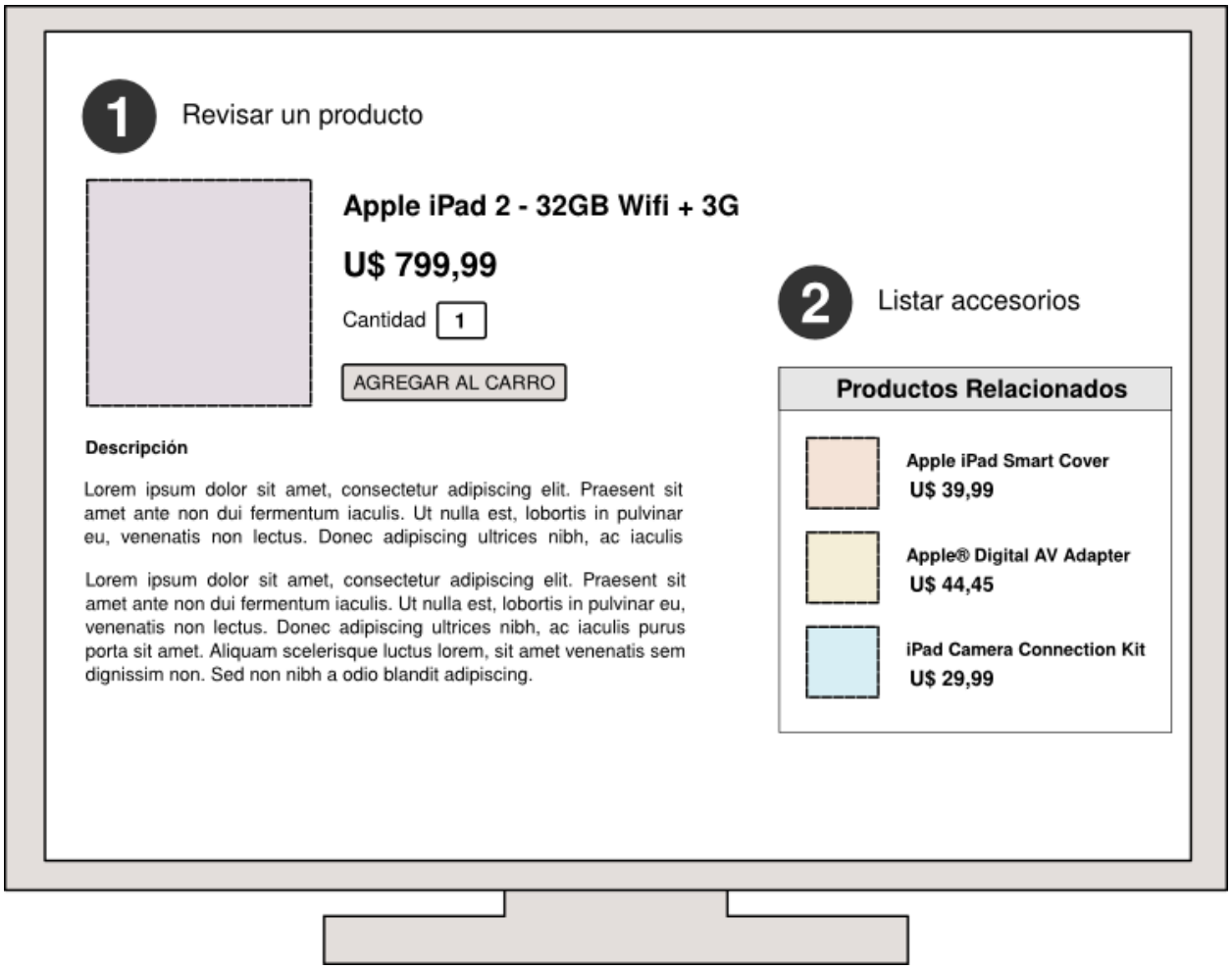


Ilustración 12. Productos Relacionados

Del lado del administrador lo que se podría hacer es utilizar una plataforma e-commerce que permita incluir productos relacionados dentro de las páginas para describir cierto producto. Esta plataforma debe permitir escoger manualmente productos relacionados y de esta manera darnos otra gran ventaja, ya que nosotros podemos ver relaciones que un programa de software no lo haría (tales como relacionar diferentes piezas de ropa para crear un conjunto).

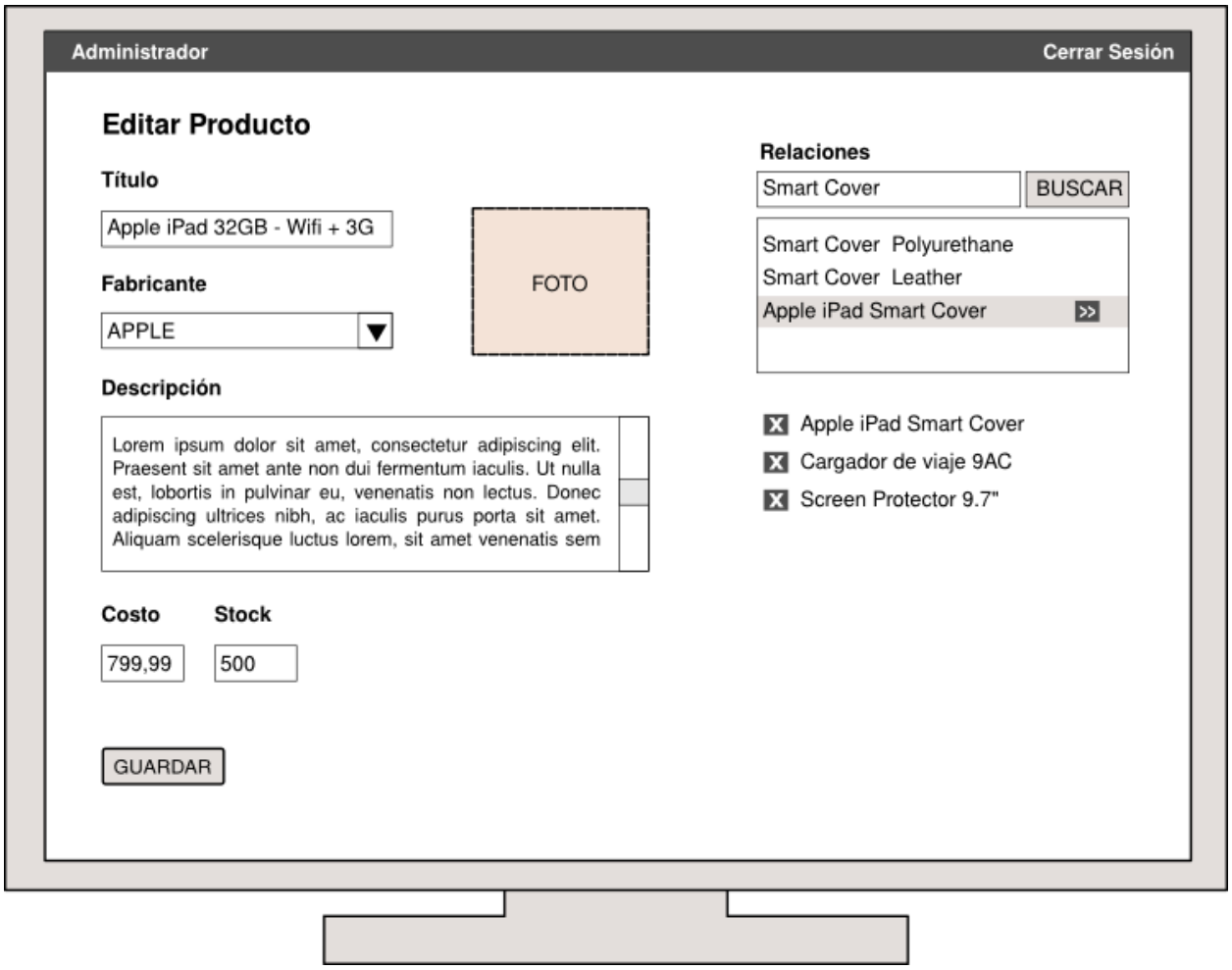


Ilustración 13. Administración de Productos Relacionados

Del lado del usuario su experiencia final se vería enriquecida al colocar los productos relacionados más importantes, posiblemente los más vendidos en primer lugar, en otra sección de la página fácilmente visible no solo mientras realiza su compra, sino incluso mientras realiza el proceso de checkout para agregarlos de manera sencilla al carro de compras, incentivando una compra de último minuto.

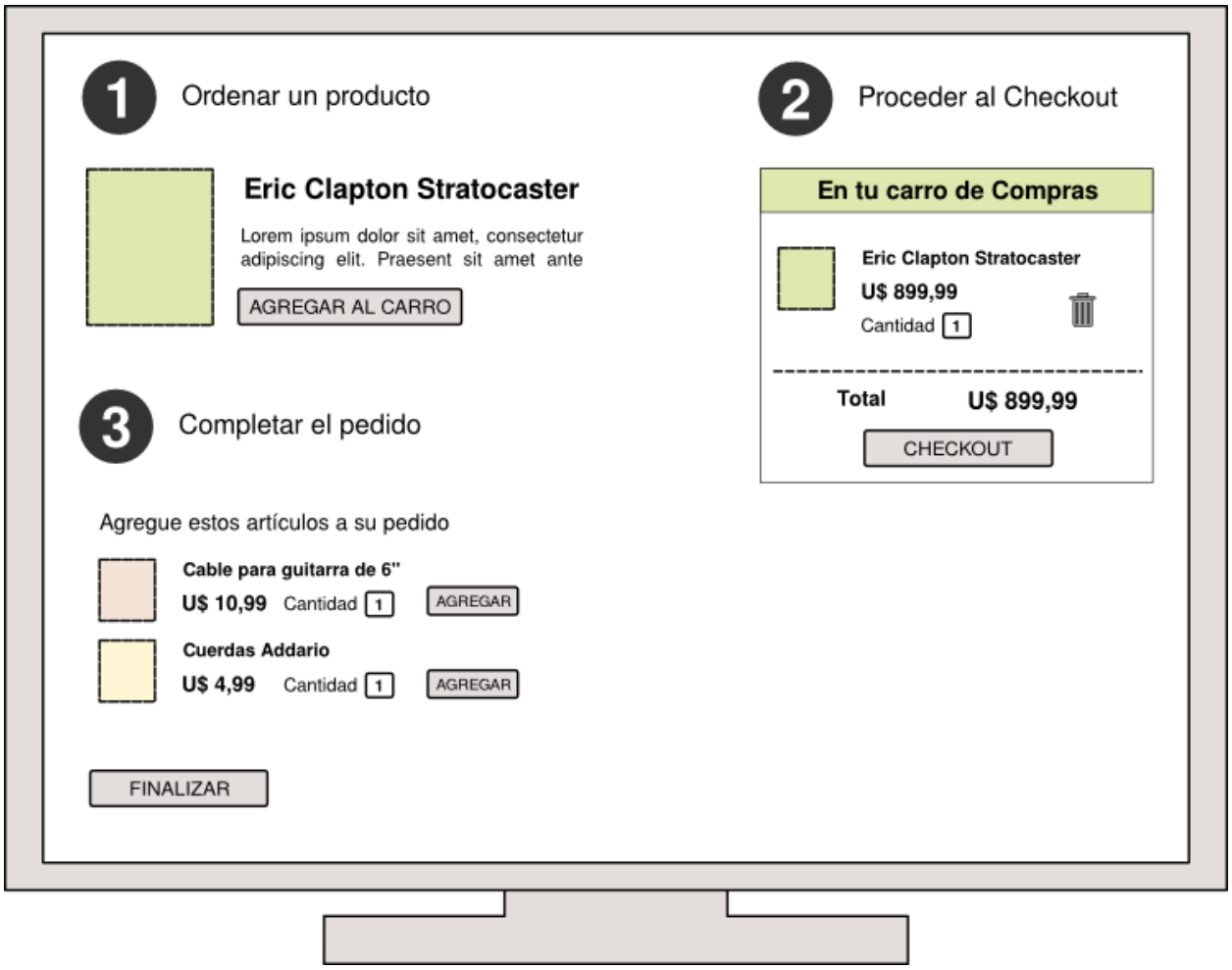


Ilustración 14. Agregar productos en el Checkout

Aquí no solo hablamos de accesorios, sino de productos similares, promociones relacionadas con el producto. Así mismo otros productos que los usuarios anteriores vieron o compraron después de comprar el primer producto. Eso da un abanico de opciones que mejora notablemente la experiencia de usuario y de paso incrementa las compras.

3. Mejorando la navegación en la Web

Las herramientas y técnicas de navegación de todo sitio Web deberían básicamente dar a los usuarios respuestas a estas 3 preguntas:

- Dónde estoy
- Dónde he estado
- A dónde puedo ir

Para explicar mejor como responder esas preguntas haremos una analogía entre los visitantes del sitio Web y los clientes de un centro comercial.

3.1 Dónde estoy

Los visitantes de un centro comercial usualmente tienen a su disposición un mapa ubicado luego de la entrada, donde pueden ver la localización de todos los locales. En ese mapa, ellos también hallarán una flecha junto a las palabras "Usted está aquí".

Ilustración 15. Donde estoy (ubicarme en el sitio)

Aunque no tan simple o directo como el símbolo de "usted está aquí", existen muchas técnicas que se pueden usar en un sitio Web para indicar a los visitantes donde se encuentran. Una de ellas es tener un conciso menú de navegación en una localización vistosa, usualmente en el margen izquierdo o en la parte superior de la página. La página actual debería ser claramente identificada de entre las otras opciones del menú, esto se puede lograr de muchas maneras:

Si se usan imágenes para las opciones del menú, se puede usar una imagen diferente para el botón que indica la página actual.

Ilustración 16. Menú de navegación (apple.com)

Si se usa HTML y CSS, se puede marcar la celda correspondiente a la página actual usando un color diferente para el fondo del contenedor, así como también cambiar el color, el peso o fuente del texto para el vínculo.

Ilustración 17. Menú de navegación (amazon.com)

Otra forma muy efectiva para dejar a los visitantes saber donde se encuentran es usar una técnica conocida como "el rastro de miga de pan" o *Bread crumbtrail* en inglés.

Ésta técnica explícitamente muestra el camino desde la página de inicio hasta la página actual, donde cada elemento del camino debería estar enlazado a su página correspondiente. Esto se muestra a continuación.

Ilustración 18. Breadcrumb en delicio.us

Cada una de estas palabras deben estar enlazadas mediante híper vínculos a sus páginas correspondientes, excepto por la última palabra debido a que corresponde a la página que ya está siendo mostrada en pantalla. Hablando en términos de CSN se podría incluir en algún lugar, posiblemente en el menú principal o en el Breadcrumb, un ícono o imagen que indique el concern actual de navegación. A su vez podría ser un enlace hacia la selección de un concern diferente.

Ilustración19. Breadcrumb y CSN

3.2 Donde he estado

En un centro comercial podríamos probablemente saber donde hemos estado con tan solo mirar atrás, o podríamos mirar nuevamente al símbolo "Usted está aquí" en el mapa e identificar los almacenes por los que hemos pasado. En un sitio Web no tenemos este lujo, sin embargo tenemos una manera simple de indicar a los visitantes donde han estado: tan simplemente dando un color diferente a los enlaces visitados.

Ilustración 20. Donde he estado (páginas visitadas con anterioridad)

El color estándar para enlaces visitados es el púrpura (como el color estándar para los enlaces no visitados es el azul). Aunque el uso de estos estándares es altamente recomendado, se puede utilizar otros colores para los enlaces. En ese caso es común practicar con una tonalidad más moderada para los visitados en contraste con los no visitados (por ejemplo, si usamos verde oscuro para los no visitados, usemos verde claro para los visitados).

Esto se puede hacer fácilmente mediante una hoja de estilos agregando las reglas necesarias para las los elementos <a> y sus pseudo clases: visited, link, entre otras.

```
a: link {  color: #007700 }  /* links no visitados*/

a:visited { color: #008800} /*links visitados*/

a:active {color: #448800} /*links activos*/

a:hover {color #888800} /*cuando el ratón pasa por encima}

a:focus {color: #558800} /*el seleccionado actualmente*/
```

Estas mismas reglas podrían incluir también imágenes si se quisiera, siempre que sea claro distinguir los enlaces ya visitados.

3.3 A dónde puedo ir

Una vez más, usando la analogía del centro comercial podemos mirar al símbolo "Usted está aquí" en el mapa y fácilmente encontrar el camino a cualquier otro almacén del centro comercial.

En un sitio Web, la mejor manera de permitir a los visitantes conocer a donde pueden ir es ofrecer un menú de navegación lo más claro posible. Algunos lineamientos que se pueden seguir son:

Agrupar opciones de navegación relacionadas

Dar a los destinos más populares la mejor localización dentro del menú de navegación, o encontrar una manera de enfatizarlos. Por ejemplo usando una fuente más notoria.

No vincular todas las secciones del sitio desde cada página del mismo. En la mayoría de casos es suficiente vincular únicamente a las secciones más relevantes y además incluir un enlace a la página inicial desde donde se puede proveer más opciones de navegación.

Mapa del sitio

Aparte del menú de navegación, podemos proveer a los visitantes un mapa del sitio, algunos usuarios preferirán utilizar esta opción en lugar de encontrar su camino a través del menú.

Buscador

Finalmente en el mundo real, algunos clientes tienden a ir directamente a la oficina de información a pedir instrucciones para llegar a algún almacén en lugar de tratar de encontrar su camino por sí mismos. En un sitio Web el equivalente a la oficina de información es un buscador. Si un sitio Web es más que simple publicidad informativa para una marca, se debe proveer un motor de búsqueda en la página inicial, la ubicación preferida es la esquina superior derecha de la página. No es necesario que en ese instante se tengan opciones avanzadas de búsqueda, basta con un campo de texto que permita ingresar palabras clave separadas por espacios. Luego al mostrarse los

resultados y si el usuario no encuentra lo que necesita, tal vez por la gran cantidad de ellos en pantalla, puede darse la posibilidad de una búsqueda avanzada con varios criterios que permita obtener resultados más refinados.

Ilustración 21. Menú con buscador en Apple.com

Debemos estar claros que aunque el contenido puede ser lo más importante del sitio, no sirve de nada si los visitantes no pueden encontrarlo. Siguiendo estos simples lineamientos de navegación brindaremos un sitio amigable al usuario y garantizaremos que los visitantes vuelvan con frecuencia.

3.4 A dónde quiero ir

La pregunta que faltaba en esta sección se relaciona CSN y con el objetivo que planteamos en un inicio, el de personalizar el esquema de navegación en base a las necesidades del usuario final.

Bien sea con una barra de buscador, con el menú principal, con el Breadcrumb o con un menú contextual (botón derecho). Los atajos o enlaces a la información deseada por el usuario, deben estar visibles y deben ser acordes al contexto, algunas ideas se describen a continuación.

Con el buscador: Al introducir palabras clave y presionar buscar, los resultados deberían estar "inteligentemente" ordenados de tal manera que se pueda acceder primero a aquellas páginas relacionadas con el concern actual.

Ilustración 22. A donde quiero ir (buscador)

Otra alternativa sería la de filtrar la información, para que solamente se muestren las páginas antes mencionadas y de esta manera se ayude a reducir el número de resultados. Sin embargo, esta última alternativa podría ocasionar que el usuario no halle lo que busca y lo confunda, sobre todo si el sistema no es claro al indicarle el concern actual.

Una solución sería únicamente destacar los enlaces a las páginas del concern de interés, tal vez en la columna principal, y colocar los enlaces a las demás páginas en una región menos visible como la columna derecha. Si el usuario elige irse por una de

estas páginas el sistema registraría un cambio de concern y el esquema de navegación se reconfiguraría.

Ilustración 23. Donde quiero ir (resultados filtrados)

Usando un menú contextual también se puede mejorar el esquema de navegación, o incluso restringir ciertas funcionalidades al usuario. Imaginemos un concern de administración al que el usuario ingresa luego de registrarse en el sistema con ciertos privilegios.

Ilustración 24. Donde quiero ir (menú contextual)

Una de las tareas más requeridas podría ser la edición de los contenidos, o tal como se muestra en la ilustración dejar la posibilidad para un cambio de concern haciendo clic derecho sobre el concern actual. Este último uso podría ser útil al administrador para observar cómo quedaría una página al ingresar en un concern en particular. Así mismo sería cómodo para el usuario final el poder cambiar a otro esquema de navegación en cualquier momento.

3.5 Otras buenas prácticas de navegación deseadas

La navegación debe estar perfectamente planeada, ya que si el visitante se pierde dentro de un sitio y no encuentra lo que está buscando lo abandonará de inmediato. Guiar a los visitantes hacia la información que requieren de forma rápida e intuitiva es el objetivo de todo sistema de navegación.

Uno de los problemas que enfrentan los sitios web cuando empiezan a crecer es que su estructura de navegación, con la que los visitantes acceden de una página a otra, queda inservible. Nunca será lo mismo un sitio pequeño con unas diez o veinte páginas que uno que contenga miles de ellas.

Es así que para cerrar con lo concerniente a problemas de navegación y las posibles soluciones que se pueden emplear, mencionamos otras ideas para enriquecer la experiencia del usuario que no debería faltar en un sitio Web profesional basado en los requerimientos del usuario.

Estructurar las Secciones por niveles

Se debe agrupar las páginas por temas siguiendo siempre una jerarquía. Hay una regla llamada "Siete Más Menos Dos", que dice que la mente humana puede retener en el corto plazo unos siete conceptos a la vez, más o menos dos, dependiendo de la capacidad intelectual de cada individuo y el tipo de información. De forma que se debe intentar que el menú principal no supere las siete secciones.

¿Qué información demandan los visitantes? Se debe agrupar la información de forma que sea lo más fácil posible para ellos encontrarla. *No se puede condicionar la navegabilidad de la web a la estructura formal de la organización*.

Utilizar títulos para cada uno de los temas, pero sin salirse de los estándares comúnmente usados en la Web, los visitantes están acostumbrados a términos como “Inicio”, “Contáctenos”, “Quiénes Somos”, "Inicio", y por lo tanto eso será lo primero que buscarán en un menú.

La manera más común de conseguir una estructura por niveles es mediante la utilización de menús desplegables o de breadcrumbs.

No usar Frames

Los frames (marcos) dividen la pantalla en dos o más áreas, y utilizan diferentes barras de desplazamiento para moverse en cada uno de ellas. La ventaja de los frames es que mantienen el menú siempre a la vista, pero crean más problemas de navegación de los que resuelven.

- El uso de frames dificulta la inclusión de una página en Favoritos.
- Una página llena de barras de desplazamiento da un aspecto muy poco estético al sitio web.
- Algunos navegadores no imprimen el contenido de los frames correctamente.
- Los frames aparecen descompuestos en monitores con poca resolución.
- Cada frame trabaja por sí solo, lo cual implica que actúa como una página diferente y trae problemas de navegación en cuanto a menús o breadcrumbs. Solo imaginemos la necesidad de colocar un menú desplegable en una frame en la parte superior de una página, éste menú al desplegarse solo se muestra en ese frame y se esconde en el resto.
- Los frames aparecen con alto y ancho fijos, por lo que si sus contenidos exceden el tamaño establecido aparecerán barras de desplazamiento, arruinando el diseño planeado.
- Por último desde el punto de vista de la programación es difícil manejar un request para una estructura con frames, como por ejemplo el submit de un formulario, dado que solo se envía la información del frame que lo contiene. Esto implica tener una programación dividida para manejar cada sección contenida en un frame diferente.

Por todo esto es preferible tener una sola página con toda la información distribuida en secciones bien establecidas. Esto se ve mejor no solo desde el punto de vista del diseño sino de la programación al brindar facilidad de mantenimiento.

Menús desplegables

Otra forma de conducir visitantes al interior de la web es poniendo un menú desplegable en la parte superior, con secciones identificables por medio de espacios o separadores. No ocupan mucho espacio y facilitan mucho la navegación. Lo malo es que los visitantes tienen que hacer "clic" sobre cada opción del menú para ver sus contenidos y mientras más niveles tenga el menú, más demorado encontrar lo que buscamos.

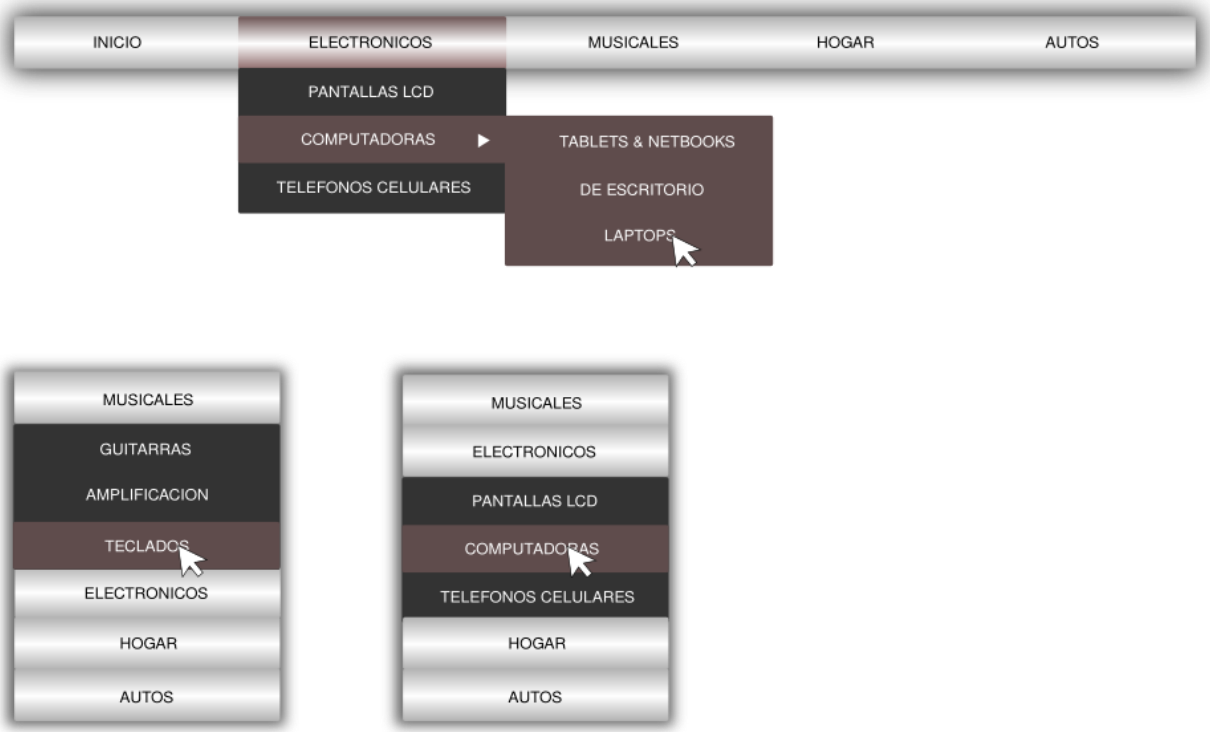


Ilustración 25. Menús desplegables y acordeón

Existe una variación de los menús desplegables que es muy común en sitios bien estructurados, conocidas como menús acordeón. Estos elementos permiten optimizar espacio al igual que los desplegables, al solo desplegar los hijos de un ítem a la vez pero éstos los hacen en forma vertical, y lo mejor de todo es que no necesitamos

abandonar la página actual al desplegar otro ítem, pues la programación necesaria para su construcción se encuentra del lado del cliente mediante Javascript. Sin embargo solo se permite tener dos niveles de anidación pues debido a su disposición vertical.

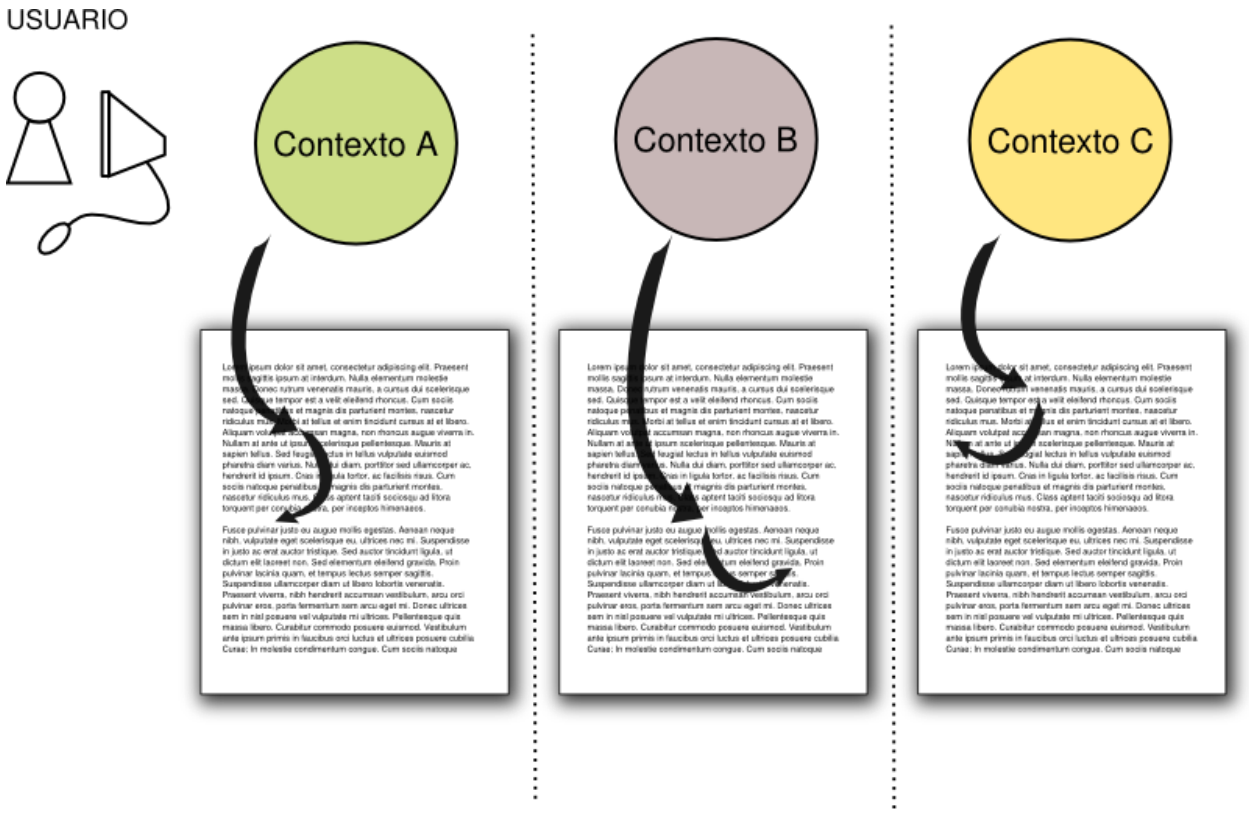
Utilizar buscadores interiores

Si los visitantes no encuentran con facilidad lo que necesitan dentro del sitio web, tal vez una mejor idea sería colocar la mayor cantidad de información dentro de una misma página. El principal problema es que si la página es muy larga, el usuario se cansa de usar la barra de desplazamiento o de leerla; en estos casos un buscador interior puede ser una solución rápida. Esta herramienta puede ser de gran utilidad para mejorar la navegabilidad de la web, y su implementación es muy sencilla.

La idea es colocar al inicio de la página enlaces hacia lugares dentro de la misma página. Esto se consigue llenando el atributo *href* del enlace con #titulo, donde titulo es el ID de un elemento colocado en el lugar de destino. Aquello produce un comportamiento similar al de un Bookmark en un archivo de texto.

Las ventajas son evidentes, pues en primer lugar no requieren recargar la página y luego hemos eliminado un nivel de menú, brindando un esquema de navegación más simple.

Se puede obtener una mejora a esta herramienta usando diferentes esquemas de navegación, al permitir que sin cambiar los contenidos de una página grande se modifiquen los atajos al inicio, para llevar al usuario a distintos lugares dentro de la misma página según sus intereses y evitando largas lecturas.



Mapa del Sitio

Otro sistema de navegación que ya se mencionó con anterioridad es el mapa del sitio, que consiste en una página de enlaces que muestra la distribución por niveles del sitio web. Es bueno incluir un enlace al mapa del sitio en cualquier menú o sección vistosa dentro de la página de inicio, ya que esta es una de las alternativas que más agrada a los visitantes para hallar y entender la estructura de un sitio.

Una vez más podemos enriquecer esta herramienta si pensamos en separar la navegación en diferentes esquemas, de tal manera que los enlaces en el mapa se muestren según el contexto de uso. Podemos mostrar y ocultar en el mapa, páginas específicas para simplificar la navegación en el sitio y permitir al usuario encontrar lo que busca en un tiempo menor.

3.6 Resumen

Los sitios Web deben centrar su atención en los usuarios que los visitan, y por lo tanto pensar su diseño en la facilidad de uso, para que sus invitados encuentren lo que buscan en el menor tiempo posible. El desarrollo Web exitoso requiere de la combinación de destrezas en conocimiento del negocio, HTML, diseño gráfico, y técnicas de usabilidad en la Web.

Se vuelve indispensable seguir buenas prácticas para una navegación ordenada, como diferenciar los enlaces visitados de los aún por visitar, incluir mapas del sitio, buenos motores de búsqueda, enlaces a páginas o artículos relacionados y sobre todo tener todo esto a la vista del usuario.

Se recomienda usar más de un esquema de navegación para que el sitio web ofrezca diferentes opciones de búsqueda a los visitantes. Se debe prestar una especial atención a los sistemas de navegación que se van a implementar en el sitio, basándonos en los requerimientos del usuario y en el contexto o escenario de uso actual.

Con respecto a enriquecer la experiencia del usuario basada en navegación sensible a separación de concerns, algunas soluciones son:

- Adaptar la presentación de un sitio Web según un contexto.
- Filtrar u ordenar los resultados de una búsqueda según el esquema actual de navegación.
- Presentar los productos o artículos relacionados, junto al detalle de otro en la misma página, dando prioridad a lo que le interesa al usuario.
- Uso de buscadores interiores a través de atajos diferentes al inicio de la página para cada esquema de navegación, lo que permite que el usuario sea llevado por determinadas paradas de interés a lo largo de textos extensos.
- Ciertos elementos que deben estar siempre a la vista, como el caso de un carro de compras, pueden variar su posición como consecuencia de una reorganización

de los elementos de una página, solicitada por un cambio de esquema de navegación.

- Mapa del sitio configurado para un contexto específico que oculte los enlaces a páginas que no sean de interés para el usuario.

Ahora que comprendemos los problemas de navegación y planteamos algunas alternativas para su solución, basadas en la utilización de distintos esquemas de navegación, debemos diseñar una arquitectura de software que permita la construcción de sitios con estas características de forma rápida y organizada. En el siguiente capítulo tratamos este aspecto a profundidad.

4. Esquemas de Navegación

4.1 Introducción

En capítulos anteriores hablamos de lo importante de tener un esquema de navegación ordenado, que permita al usuario encontrar fácilmente su camino a través de cada página y del sitio. Nos dimos cuenta, luego de analizar los problemas que son frecuentes en algunos sitios Web que la solución caía tanto en la buena distribución de los contenidos, como en una agrupación lógica de los enlaces que nos permiten navegarlos; y que además la separación y selección de los esquemas dentro de un contexto dado podía enriquecer la experiencia del usuario final.

Existen diversas maneras de lograr esto y algunos sitios en Internet ya se han hecho esfuerzos alcanzando resultados aceptables. Sin embargo no existe una arquitectura de soporte bien definida que permita construir estas páginas en forma ordenada, sino más bien se ha preferido en agregar al código existente estructuras condicionales que dirijan el flujo de la aplicación según el caso. Esta forma de hacer las cosas, no solo puede ocasionar errores al editar código ya probado con anterioridad, sino que dificulta su mantenimiento y lo vuelve confuso para otro programador.

Existe una tendencia actual para utilizar arquitecturas que faciliten la modularidad del software, mediante la correcta separación de concerns o aspectos. Esto consiste en que exista una clase independiente o un conjunto de clases agrupadas en módulos, que son creados por separado para resolver un aspecto específico de la aplicación y que interactúan con otros módulos al momento de unirlos. Esto habilita a comunidades de programadores a que puedan trabajar de forma independiente, una vez que se hayan establecido los estándares de programación y documentación que garanticen que las partes funcionen correctamente cuando sean acopladas al núcleo de la aplicación.

Utilizando esta misma metodología se pueden conseguir diferentes esquemas de navegación, teniendo uno para cada concern planteado luego de un previo análisis realizado a comienzos del desarrollo. Cada esquema se define y evoluciona de manera

independiente lo que garantiza su fácil mantenimiento en una etapa posterior.

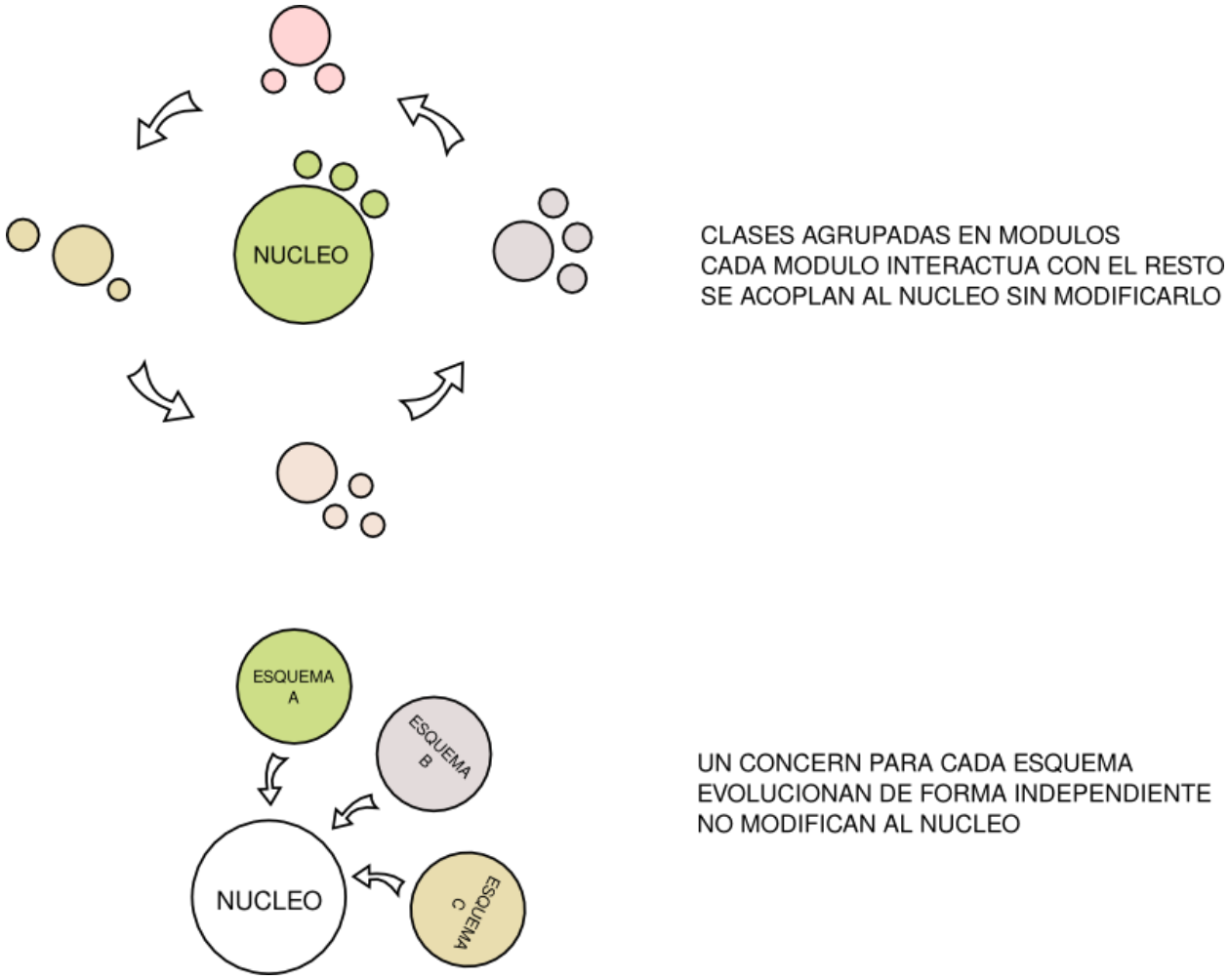


Ilustración 26. Esquemas de navegación y Arquitectura modular

Así, si en algún momento vemos la necesidad de agregar otro esquema de navegación, no necesitamos modificar el código ya existente o depender de la implementación de un esquema anterior, excepto en el caso de que existan sub esquemas hijos del mismo padre. Necesitamos primero entender cómo funciona la separación de concerns y que alternativas existen para su implementación. Este tema se trata en el siguiente apartado.

4.2 Separación de Concerns

4.2.1 Programación usando niveles de Abstracción

En los 70s, científicos de computación trataban de resolver problemas complicados de arquitectura de software originados de las necesidades de cada día en reutilizar y mantener software. En 1968 se introduce por primera vez el concepto de nivel de abstracción, el cual es usado para hacer que el diseño de código a nivel de sistema operativo sea más fácil.

Es decir un nivel de abstracción k es un elemento en la jerarquía de un sistema de software que consiste en un grupo de módulos cuya implementación llama a módulos de nivel k -1, donde el nivel de abstracción 0 es considerado ya sea instrucciones a nivel de hardware o la implementación de un lenguaje de programación.

Generalmente hablando, todo el software es desarrollado y evolucionado de acuerdo a este principio, y cada programador de software empieza en un nivel de abstracción lo suficientemente alto, hasta llegar al 0.

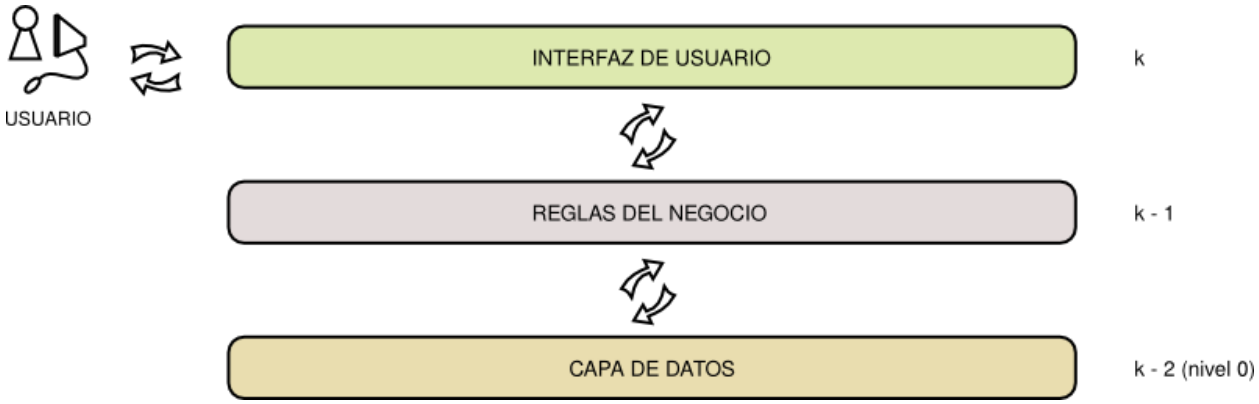


Ilustración 27. Separación de concerns en niveles de abstracción

De igual manera, el usuario de la aplicación únicamente interactúa con el nivel de abstracción más alto y el flujo se encamina hacia abajo hasta llegar a la capa más baja, o hasta donde el requerimiento pueda ser satisfecho, allí se obtendrá un resultado que viajará nuevamente camino arriba hasta mostrarse al usuario que hizo la petición.

El número de niveles puede variar y depende del desarrollador definir los aspectos a tratar en cada nivel. Generalmente se tiene una estructura muy parecida a la que se mostró en la Ilustración 27, y de allí en adelante se empieza a subdividir los niveles para reducir la complejidad en la programación.

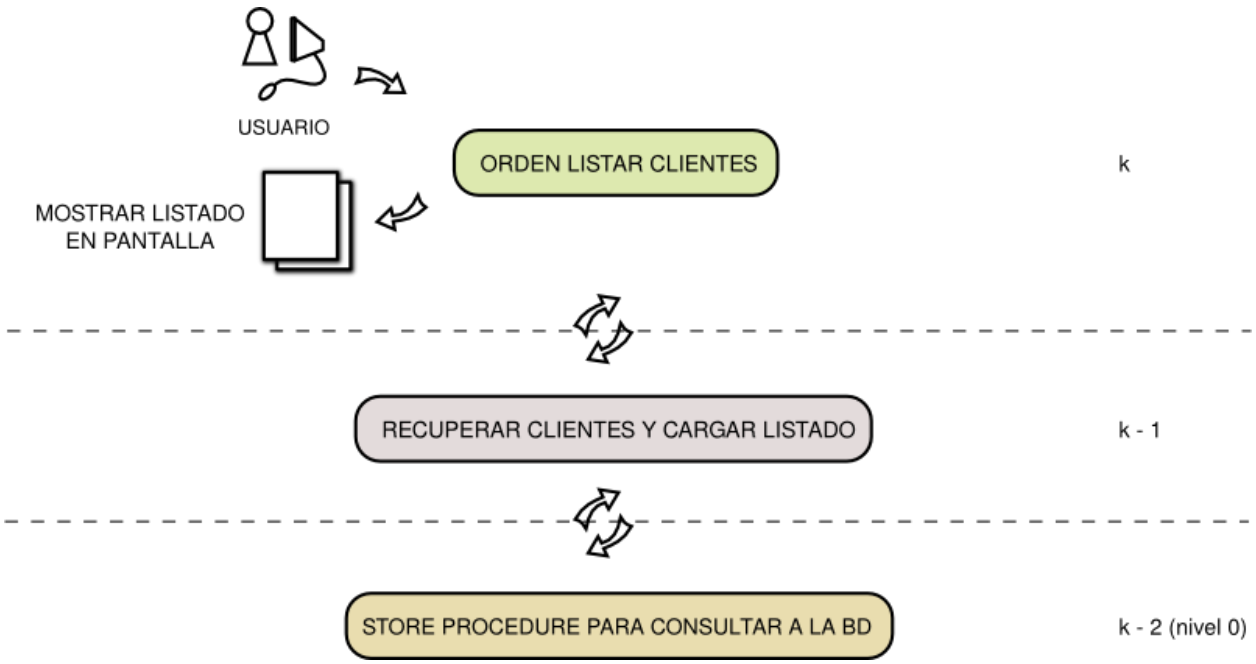


Ilustración 28. Flujo del programa a través de niveles de Abstracción

Sin embargo tiene también sus desventajas a la hora de trabajar en equipo, ya que si bien los aspectos han sido separados en niveles, cada capa depende directamente de la capa inferior, por lo que si la capa de negocios necesitara recuperar de la base de

datos un listado de clientes bajo un criterio dado, se tendría que coordinar y esperar que la persona responsable de la capa de datos defina e implemente esta funcionalidad.

Según la ingeniería de requerimientos orientada a concerns, un concern de aplicación es definido como cualquier conjunto coherente de requerimientos, los cuales se refieren a un tema o comportamiento particular de la aplicación. En algunos frameworks los encontramos en forma de módulos bien delimitados, que tratan un aspecto concreto en la construcción de las aplicaciones. Los concerns pueden también reflejar aspectos no funcionales de la aplicación, es decir cualquier asunto de consideración en el sistema de software.

4.2.2 Programación Orientada a Aspectos

En los casos más simples, un sistema de software puede ser diseñado e implementado como un conjunto de módulos organizados de acuerdo al esquema jerárquico de nivel de abstracción. Sin embargo, luego se concluyó que dicho mecanismo arquitectónico básico no era suficiente debido a que existían muchos otros aspectosa tratar (concerns o tareas, sub tareas, ideas).

Estos concerns no podían simplemente ser implementados como una jerarquía de módulos, los programadores frecuentemente necesitaban "atravesar" (cross – cut) la jerarquía para agregar nuevas características o para modificar las existentes. De allí aparece el concepto de "cross – cuttingconcern" introducido en programación orientada a aspectos, y definido como el Concern que no puede ser implementado por un solo procedimiento generalizado (o conjunto relacionado de módulos).

El ejemplo más simple de un cross – cuttingconcern es el logging, en el que se requiere registrar mensajes en un archivo log antes o después de la ejecución de un evento. Esto require incluir escrituras en el log a lo largo del código y muy prrbablemente para más de un nivel de abstracción. Para un sistema de software, esto nos lleva a pensar la estructura incluyendo otra dimensión: la vertical.

A continuación se muestra la arquitectura de un sistema de software de dos dimensiones expresada como una matriz rectangular, o también vista como una jerarquía de niveles de abstracción con un conjunto de cortes verticales.

	Logging	Actualización de Vistas
Interfaz de Usuario	x	x
Reglas del Negocio	x	x
Datos	X	x

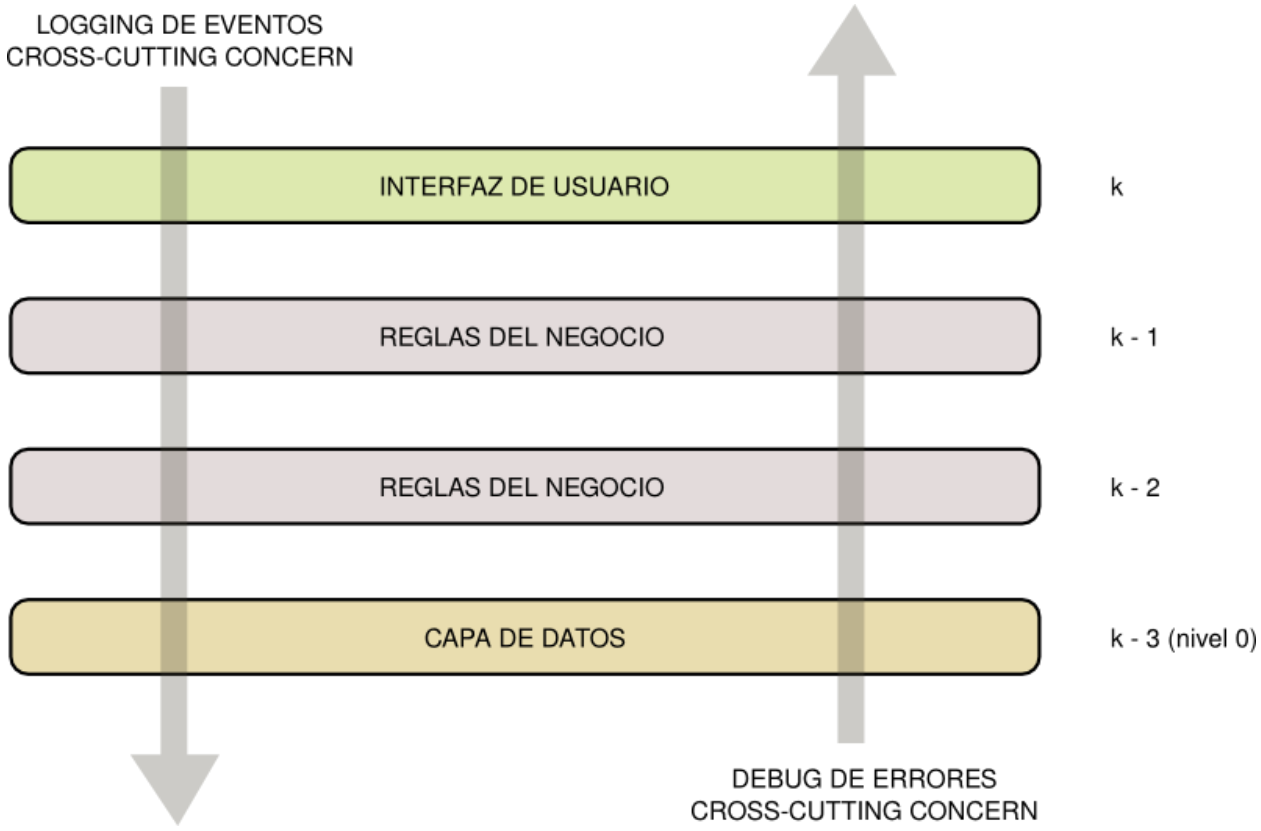


Ilustración 29. Separación de concerns y Cross-cuttingconcerns

La programación orientada a aspectos es un nuevo paradigma de programación que aumenta la modularidad permitiendo la separación de cross – cutting concerns. Esta técnica permite extender una aplicación haciendo que los concerns propios de la misma (coreconcerns) no necesiten ser editados, es decir no se debe modificar el código existente.

En lugar de crear un nuevo conjunto de clases y embeber su uso llamando a sus métodos desde cada clase original, "tejemos" los diferentes aspectos nuevos al código original al momento de la compilación. Las ilustraciones posteriores explican lo que sucede al usar esta técnica.

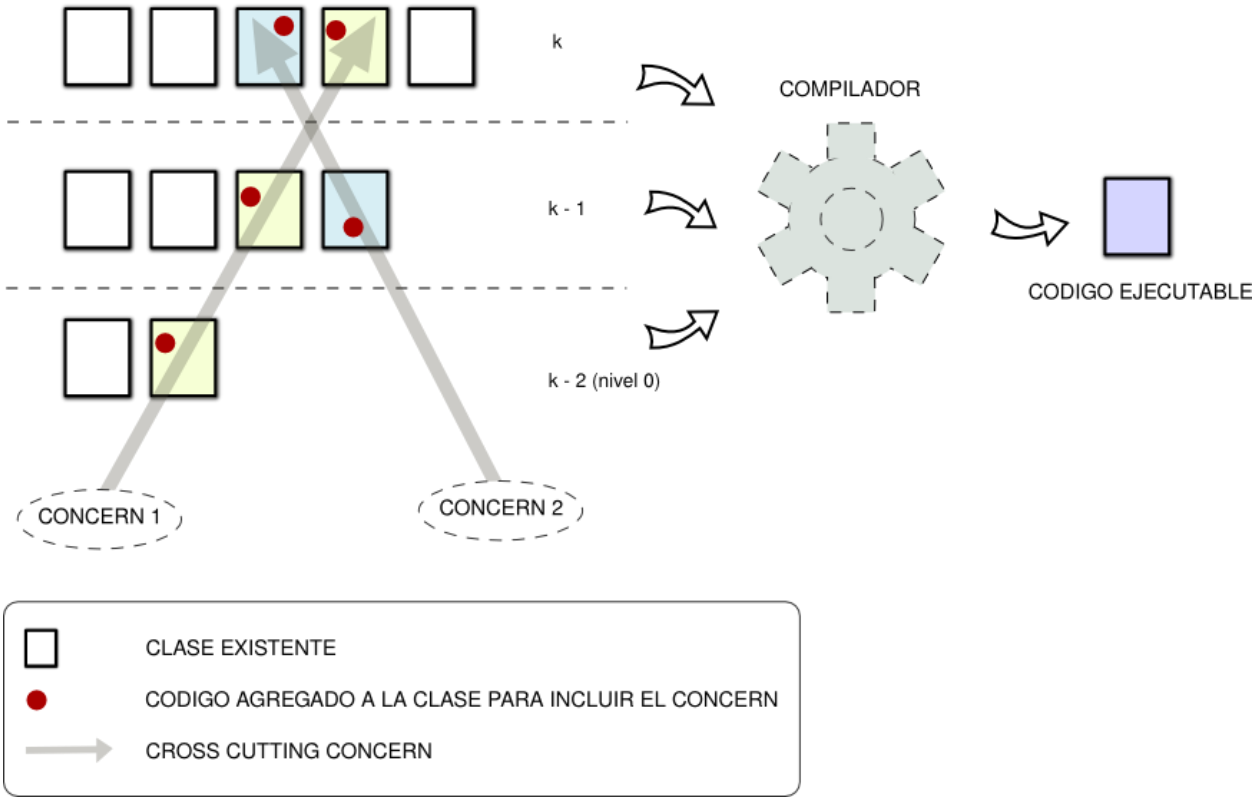


Ilustración 30. Programación sin AOP

El proceso normal para la mayoría de lenguajes de programación antes de la ejecución de un programa, involucra la compilación de las clases y código para la generación de

código que pueda ser directamente ejecutado bien sea por el sistema operativo o por una máquina virtual.

Los crosscutting concerns a pesar de estar implementados en clases diferentes deben invocar a sus métodos desde las clases cliente, lo que añade código a la aplicación en varios de los niveles e incluso en diferentes lugares dentro de una misma clase o función. Este problema parecería inevitable a no ser por la programación orientada a aspectos que nos permite mantener el código original limpio a la vista del programador.

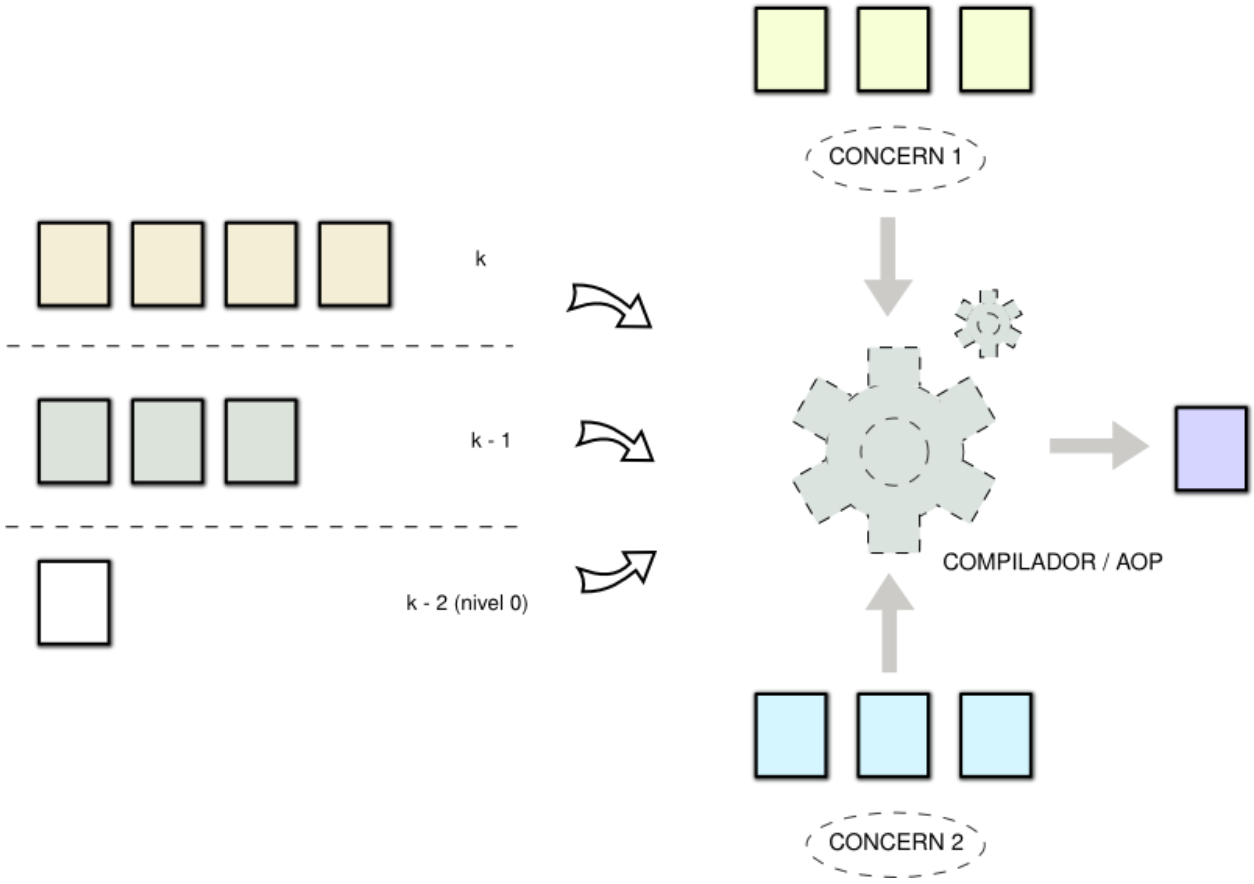


Ilustración 31. Programación con AOP

La AOP permite agregar y en cierto modo modificar una clase existente en otra zona de la aplicación, pero sin realmente alterar de manera visible las líneas de código de esa clase, sino que fusiona los aspectos antes de la generación del código ejecutable. No todos los lenguajes de programación permiten AOP y en muchos casos su implementación no es tan sencilla de lograr.

Una programación modular propiamente dicha debería mantener cada aspecto por separado, y aún así dejar que las partes interactúen entre sí. Sin una arquitectura o tecnología que soporte esta particularidad, esto no sería posible. Recordemos que el desarrollador de un **cross – cutting concern** y por lo tanto de un aspecto, tiene que inyectar fragmentos de código dentro de otro código que implementa a otros concerns desarrollados por otros programadores y al que el desarrollador puede no estar familiarizado. Así que en cierto modo AOP puede ser vista como la tecnología de software que ayuda a separar módulos escritos por otros desarrolladores.

4.3 Navegación sensible a concerns (CSN)

En apartados anteriores hablamos de esquemas de navegación y el papel importante que juegan en enriquecer la experiencia del usuario Web final. También dimos una explicación de la separación de concerns y su presencia en el desarrollo de aplicaciones para mejorar la modularidad, y en consecuencia el mantenimiento y evolución de las mismas.

Si bien estamos convencidos que debemos considerar tanto una buena navegación, como una correcta arquitectura al momento de construir aplicaciones Web, parecería que son trabajos separados realizados en diferentes etapas y tal vez por diferentes personas encargadas. Sin embargo la principal motivación de CSN es presentar que mediante la separación de concerns de navegación, la experiencia del usuario final se verá enriquecida; por lo tanto su uso ya no se limita solamente a organizar la programación.

Este trabajo apunta a mejorar el acceso cognitivo a la información, lo que significa proveer al usuario con la información necesaria en cada concern, y que esté organizada y presentada de manera más oportuna.

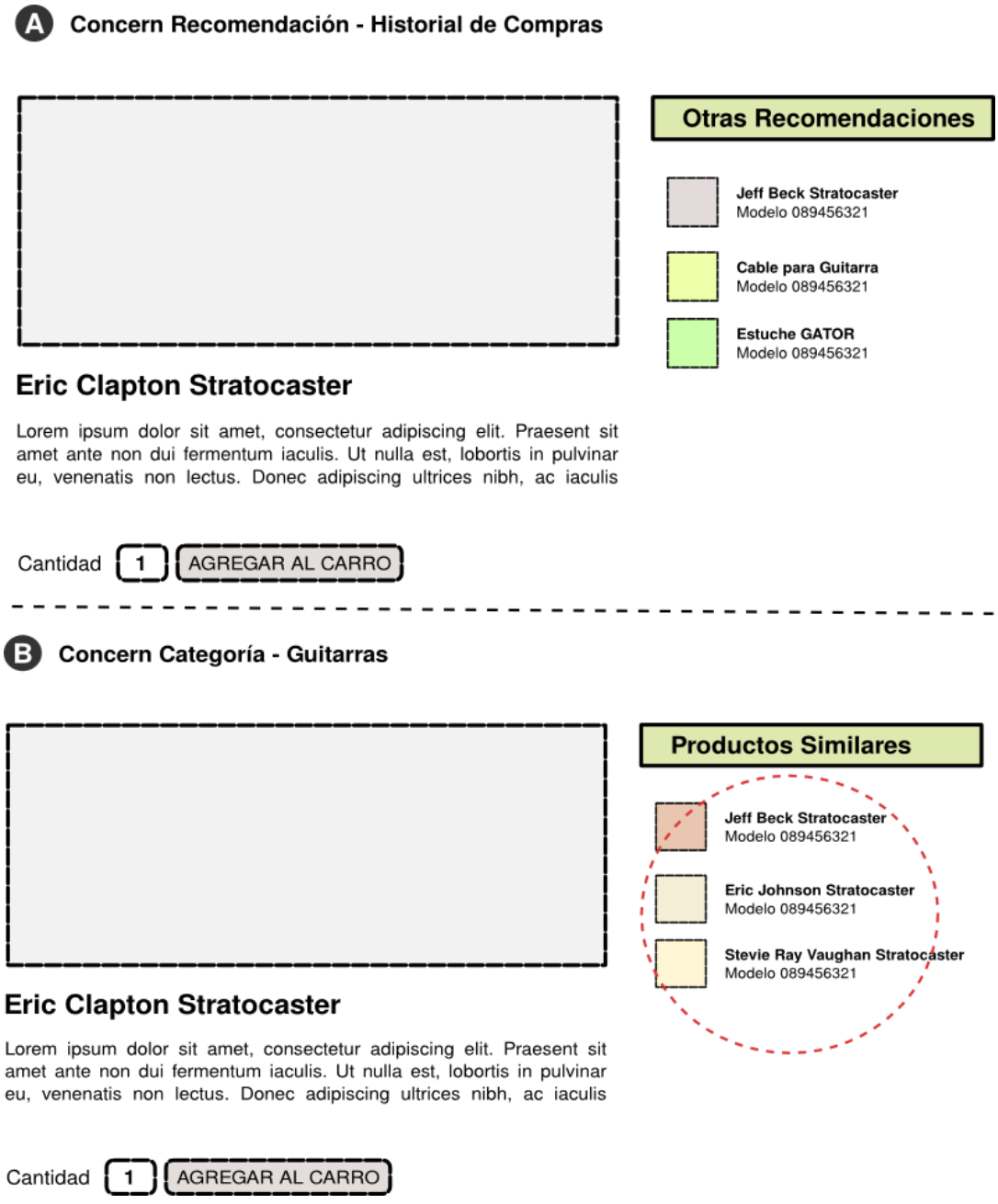


Ilustración 32. Diferentes concerns de navegación

Esta página tiene la misma estructura a pesar de que el usuario pudo haber llegado a ella al seleccionar una categoría, o como recomendación del sitio a partir de su historial de compra, o desde el carro de compras para revisar el producto antes del checkout, entre otras posibilidades. Esta estructura deficiente en la que un objeto se ve igual sin importar el contexto desde el que es accedido, disminuye la usabilidad y dificulta el mantenimiento.

De ahora en adelante vamos a llamar **nodo** a todo objeto navegacional, como puede ser una página Web o una región dentro de la página, al que podemos acceder desde una URL o viniendo desde otro nodo. De igual manera vamos a llamar **rol** al comportamiento que se ata a un nodo para indicar la información adicional, los enlaces u operaciones que se mostrarán en el concern correspondiente.

En el apartado referente a separación de concerns habíamos definido un concern como cualquier aspecto relevante en un sistema de software. Más concretamente un concern de navegación es aquel que como su nombre lo indica afecta la navegación y la estructura de esta a lo largo de la aplicación; es decir los contenidos presentados y sus enlaces, por lo tanto tiene impacto en la manera en que los usuarios navegan la aplicación.

Cuando alguien navega por la Web, lo hace con una tarea específica en mente; las metodologías de diseño para la Web plasman estos requerimientos, especificados como casos de uso, en modelos conceptuales y navegacionales. CSN complementa estas ideas mediante estrategias para enriquecer los objetos de navegación con información específica al concern que el usuario atraviesa. Así los mecanismos de CSN y su alcance deben ser definidos por el diseñador en base a las necesidades del usuario.

Parte de observar en los sitios actuales que cuando llegamos a una página, esta luce exactamente igual independientemente de la razón por la cual el usuario llegó a ella. Pongamos como ejemplo un sitio de compras en línea, en el cual el usuario llega a una página para ver la descripción de un producto, ésta situación ya la estudiamos cuando analizamos los problemas y buenas prácticas de navegación. Una muestra de este

enriquecimiento en base al contexto actual de un sitio Web se muestra con un ejemplo en la Ilustración 32.

4.3.1 Enriquecimiento de Concerns de Navegación

Los concerns que caen dentro del dominio referente a la navegación son aquellos que pueden ser expuestos al usuario, identificamos las siguientes categorías:

Concerns de tarea.- Los relacionados con acciones de alto nivel que el usuario puede realizar en una aplicación Web.

Concerns de Tópico o temáticos.- Cuando se realiza una acción de búsqueda en un sitio los temas, áreas o categorías que agrupan a las páginas se convierten en una clase de concerns.

Concerns de navegación puros.- Son esquemas de navegación predefinidos como tours guiados o conjuntos que involucran secuencias de páginas.

Estos concerns se pueden enriquecer para mejorar la experiencia del usuario mediante los siguientes atributos y características:

- Nuevos contenidos y/o actualizados
- Enlaces y anclas específicas
- Nuevas Operaciones

Del listado anterior se derivan los siguientes patrones para cada categoría de concerns de navegación, los cuales se repiten para la mayoría de sitios Web:

Para los concerns de tarea.-Cuando se trata de un proceso de negocio, como en los sitios e-commerce se puede agregar nuevas operaciones para aumentar la usabilidad, así mismo se deben remover aquellas que puedan ocasionar errores en el proceso, o al menos agregar advertencias.

Para los concerns temáticos.- Se puede agregar nueva información o links específicos al tópico relacionado con el nodo o página actual. Como en el ejemplo de los productos relacionados para listar más dentro de la misma categoría.

Para los concerns de navegación puros.- Es esencial enriquecer el nodo con enlaces al índice del conjunto, y con vínculos al nodo siguiente y anterior. También se puede encontrar este enriquecimiento en navegación basada en tags, donde se incluyen enlaces a nodos con la misma etiqueta.

El diseño de la arquitectura necesaria para implementar estos enriquecimientos en base a CSN se verá más adelante en el capítulo 6, en el cual se detalla la creación de un framework en base a nuestro análisis de capítulos anteriores para generar este tipo de aplicaciones Web que soporte enriquecimiento con navegación sensible a concerns. El diseño de navegación sensible a concerns puede verse como una matriz de nodos y concerns, donde se especifica el enriquecimiento para cada caso.

4.4 Trabajos Relacionados

La separación de concerns ha dado un aporte significativo a la Ingeniería de software al simplificar el crecimiento y mantenimiento de las aplicaciones a través de la modularidad. Es una práctica muy común en aplicaciones Web actuales como CMSs donde al inicio se tiene únicamente código esencial, que contiene el núcleo del programa con las funcionalidades más básicas, y que va creciendo al irse incorporando módulos desarrollados por terceras personas de manera totalmente independiente aunque respetando los estándares acordados.

Estos módulos se relacionan con el núcleo y con los otros módulos, interactuando unos con otros sin esfuerzo debido a la arquitectura interna pensada desde un inicio para soportar esta separación de aspectos, lo cual permite el crecimiento vertiginoso de la aplicación por involucrar a programadores de todas partes.

Toda aproximación en la ingeniería Web reconoce la necesidad de plantear un diseño en componentes separados, los cuales tratan con aspectos claramente diferentes como la navegación, la presentación, la persistencia, el control de errores, los procesos de negocio, entre otros. La principal diferencia entre otros trabajos realizados, es que CSN usa una separación vertical de concerns relacionados con la esencia de la aplicación, con el fin de conseguir mejores estructuras navegacionales y no solo facilitar el mantenimiento, esto permite ver resultados a nivel de usuario y no solo a nivel de desarrollador.

Mientras se manipulan técnicas de separación de concerns ampliamente usadas, como una arquitectura MVC, AOP, o programación en capas, se recolecta información a nivel de requerimientos (etapa de análisis) para mejorar los enlaces percibidos por el usuario mientras se navega en el contexto de un concern de interés.

Estas mejoras con respecto a las estructuras planas de navegación, no conllevan a requerimientos adicionales tales como elaborar reglas o algoritmos de inteligencia artificial complicados, aunque un trabajo futuro podría llevar a la incorporación de estas técnicas más avanzadas, sino que se usan prácticas bien conocidas como la separación de concerns para generar una mejor experiencia navegacional. Además la naturaleza modular de nuestros diseños, permite agregar nuevos concerns con facilidad.Vamos a describir algunos de los proyectos existentes que tienen puntos en común con CSN.

4.4.1 Ayuda sensible al contexto

Es una clase de ayuda en línea que se obtiene para un estado específico del programa, brindando ayuda para la situación que está asociada con ese estado.

La ayuda sensible al contexto, al contrario de la ayuda en línea tradicional o de los manuales en línea, no necesita ser cargada en su totalidad para ser usada. Sin embargo cada tema debe describir extensamente un estado, situación o característica del software.

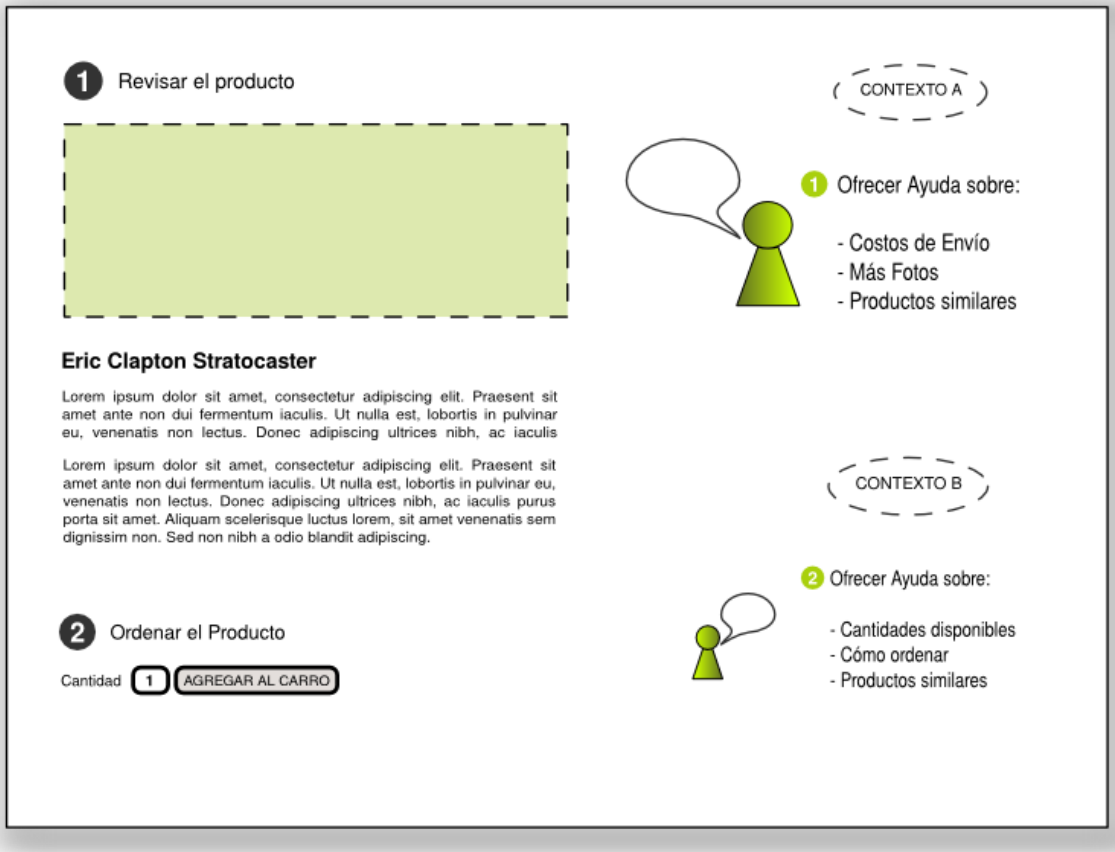


Ilustración 33. Ayuda sensible al contexto

Puede ser implementada mediante **tooltips,** mediante ayudantes virtuales o haciendo clic sobre botones específicos; la información puede desplegarse inmediatamente o luego de que el puntero del ratón se transforma en un signo de pregunta.

Ejemplos de este tipo de ayuda son:

- WinHelp de Microsoft
- JavaHelp de Sun
- INF Help de OS/2

4.4.2 Asistente de Navegación sensible a los Estados

Este trabajo no es una implementación directa para aplicaciones Web, pero dirige sus esfuerzos hacia la navegación que se adapta a los intereses del usuario.

Aquí se refiere a métodos mecánicos para asistir al usuario, el cual usa un dispositivo a control remoto para manipular una aplicación. El cerebro del asistente denominado la lógica de navegación, determina cual debería ser el estado actual de interacción entre el usuario y una aplicación; definiendo las teclas del dispositivo de control remoto que son relevantes en un estado determinado.

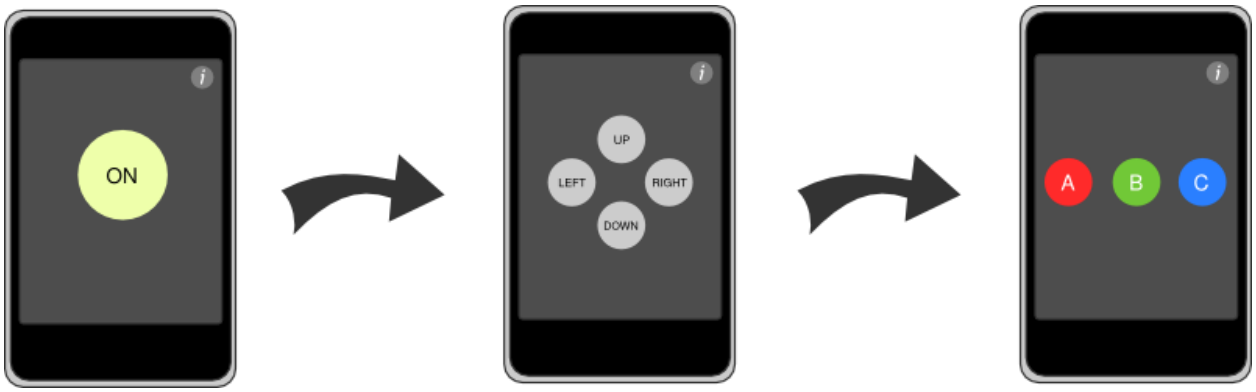


Ilustración 34. Asistente de Navegación por Estados

El asistente re ensambla la configuración de las teclas físicas asociadas dentro del dispositivo a control remoto, para de esta manera asistir al usuario dando énfasis a las acciones más relevantes, y dando menor prioridad a otros mecanismos de entrada del dispositivo que no están relacionados con la interacción definida en ese momento entre el usuario y la aplicación (omitiéndolos de la representación).

Una implementación más sencilla podría realizarse con dispositivos touchscreen que presenten diferentes configuraciones de botones para un contexto específico, evitando usar así dispositivos mecánicos para presentar y esconder las teclas.

4.4.3 Método de Diseño Hipermedia Orientado a Objetos (OOHDM)

Esta es una metodología para el desarrollo de aplicaciones Web orientada a objetos, y con principal énfasis en la navegación y en el contexto en el que navegará el usuario final.

Sus principios básicos son los siguientes:

- Contempla los objetos que representan la navegación como vistas de los objetos detallados en el modelo conceptual.
- El uso de abstracciones apropiadas para organizar el espacio de la navegación, con la introducción de contextos de navegación.
- La separación de las características de interfaz de las características de la navegación.

A grandes rasgos la metodología incluye las siguientes etapas:

- Definición del modelo conceptual, para construir el esquema conceptual que representa a los objetos, sus relaciones y colaboraciones, las cuales existen en el dominio del tema de la aplicación.
- Definición de un diseño navegacional, donde se introducirá el concepto de contextos de navegación, o espacios de la navegación. Aquí se decide cuales son los objetos navegados que pueden parecer diferentes dependiendo del contexto en el que ellos son visitados, y se debe especificar esas diferencias claramente.
- Definición del diseño de interfaz abstracta para definir aspectos de la interfaz. Esto significa definir la manera en que diferentes objetos de navegación aparecerán. Una separación ordenada entre ambos concerns de navegación y diseño de interfaz abstracta, permite construir interfaces diferentes para el mismo modelo de navegación.
- Finalmente la implementación del diseño.

CSN tiene algunos puntos en común con esta metodología, ya que ambos buscan mejorar la navegación del usuario tomando en cuenta el perfil y necesidades del mismo. Además en una aplicación de hipermedia adaptiva, los nodos y enlaces varían de acuerdo a características del usuario, su historia de navegación, etc.

Otro trabajo que encaja con estos puntos es el sucesor de OOHDM llamado SHDM o método de diseño hipermedia semántico, del cual no hablaremos en este documento pero que se recomienda investigar a partir de las referencias presentadas al final.

5. Posibles usos de CSN para aplicaciones Web 2.0

Anteriormente ya hablamos de algunas de las mejoras en la navegación que se pueden hacer, sobre todo en sitios de tipo e-commerce, ahora nos vamos a dedicara encontrar enriquecimientos con CSN para otros tipos de aplicaciones Web. Vamos a prestar especial atención a la parte de gestores de contenidos, ya que uno de los objetivos de este proyecto es el desarrollo de un prototipo para esta clase de aplicaciones.

5.1 Gestores de contenidos

Un Sistema gestor de contenidos (*Content Management System* en inglés, abreviado CMS) es una aplicación Web para la creación y actualización de contenidos de sitios Web sobre todo informativos, cuyos responsables no son los programadores como sucedía en sitios estáticos, sino que los usuarios son los administradores, el cliente o incluso los mismos visitantes del sitio.

El sistema permite manipular de manera independiente los contenidos de cada página del sitio Web, sirviendo de interfaz para facilitar la actualización de una o varias bases de datos donde se alojan estos contenidos. Varios usuarios editores pueden trabajar al mismo tiempo en diferentes secciones del Sitio según los permisos asignados, y tenerlo siempre actualizado.

Con esta herramienta, es posible darle en cualquier momento un contenido y organización distintos al sitio sin tener que modificar el HTML de cada página, debido a que la misma es generada de manera dinámica al momento de ser solicitada por un navegador; es decir que si se pide información sobre la compañía propietaria del sitio, no existe un archivo HTML independiente alojado en el servidor, sino que en ese instante se consulta la base de datos mediante un lenguaje de programación para construir la página y se la envía al navegador para su lectura por parte del usuario que la solicitó.

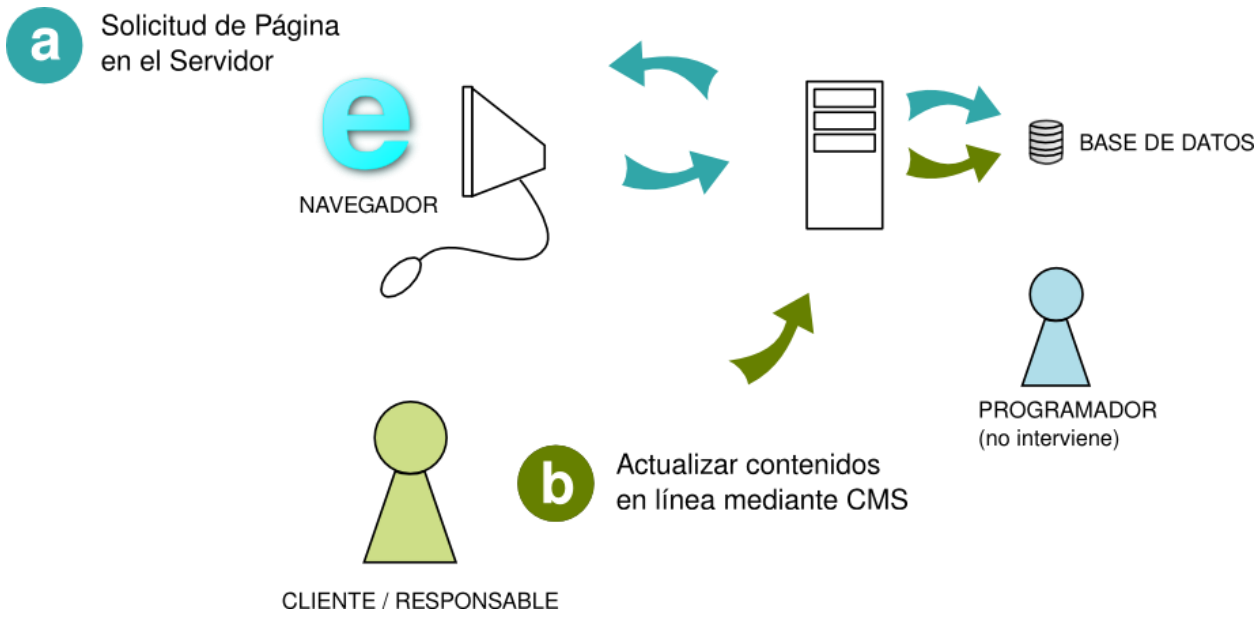


Ilustración 35. Funcionamiento de un CMS

Se debe tomar en cuenta que parte de su éxito se debe a que son fáciles de aprender y usar, por lo tanto los desarrolladores solo intervienen en la implantación del sitio y hasta su publicación, entonces es tarea de los usuarios comunes sin conocimientos técnicos el mantenimiento y actualización del mismo.

Otra característica importante son los diferentes niveles de acceso para los usuarios, partiendo desde el administrador del portal hasta un usuario final anónimo sin permiso de edición. Dependiendo de la aplicación podría haber varios permisos intermedios que permitan la edición del contenido, la supervisión e incluso la edición del contenido creado por otros usuarios, etc.

El CMS controla y ayuda a controlar cada paso en este proceso, incluyendo las labores técnicas de publicar los documentos a uno o más sitios. Sin embargo en algunos sitios con CMS una sola persona hace el papel de creador y editor, como es el caso de los blogs de autor.

Una evolución de los CMSs hacia la Web 2.0, es la administración de los contenidos de un sitio por parte de toda una comunidad de usuarios, y ya no únicamente por los administradores o los clientes, un ejemplo de este tipo de gestores son las wikis, los blogs de compañías y las redes sociales. Dependiendo del uso que se le dé a un CMS podemos separarlos en las siguientes categorías:

Plataformas de gestión de contenidos

- CMS paraForos
- CMS para Blogs
- CMS paragalerías
- CMS paraGalerías de Arte
- CMS para Wikis
- CMS paraeCommerce
- CMS para groupware

Ejemplos

Entre los más conocidos al momento de la realización de este documento tenemos los siguientes:

- CmsMadeSimple
- Drupal
- Joomla
- Mambo
- OsCommerce
- Zen Cart
- WordPress
- Al Fresco
- DotNetnuke
- Share Point de Microsoft

5.2 Agregando CSN a un CMS

Los sistemas de gestión de contenidos son generalmente utilizados para crear sitios web de carácter informativo desde sitios bibliográficos hasta revistas o diarios en línea (Ej. lanacion.com). Tanto la parte pública que presenta la información que será visitada por los usuarios comunes, como la sección administrativa para cargar los contenidos por parte del cliente, pueden verse enriquecidas agregando las bondades de CSN.

Si tomamos como referencia a sitios de noticieros, una noticia podría presentarse de forma distinta dependiendo del contexto desde el cual fue accedida para su lectura. Así mismo las operaciones de administración podrían ser diferentes en cada contexto, para conseguir una aplicación de edición de contenidos más amigable al usuario que la utiliza.

Examinemos por el momento solo el primer caso de la parte pública.A continuación damos algunos ejemplos de los concerns a considerar:

Tiempo: Al acceder a una noticia directamente desde la página inicial (home), se puede mostrar un listado de las noticias que ocurrieron en el mismo día ordenadas de la más reciente a la más antigua. Es decir sería un listado de últimas noticias y podría ser el concern por defecto al entrar al sitio.

Categoría: al acceder a una noticia desde una sección en particular (Ej. deportes), se debería dar prioridad en el listado de noticias relacionadas, a todas las noticias que tratan de deportes ya que sería lo primero que quisiera ver el usuario.

De igual manera, una misma noticia podría encontrarse dentro de varias categorías, por lo que en el listado de relacionadas luego de mostrar otras noticias para la categoría actualmente seleccionada, se presentarían en segundo lugar las de las demás categorías de la noticia. Además podría mostrarse en otra sección, un listado de todas las categorías en las que la noticia aparece de manera que el usuario pueda navegar siempre por contenidos de su interés.

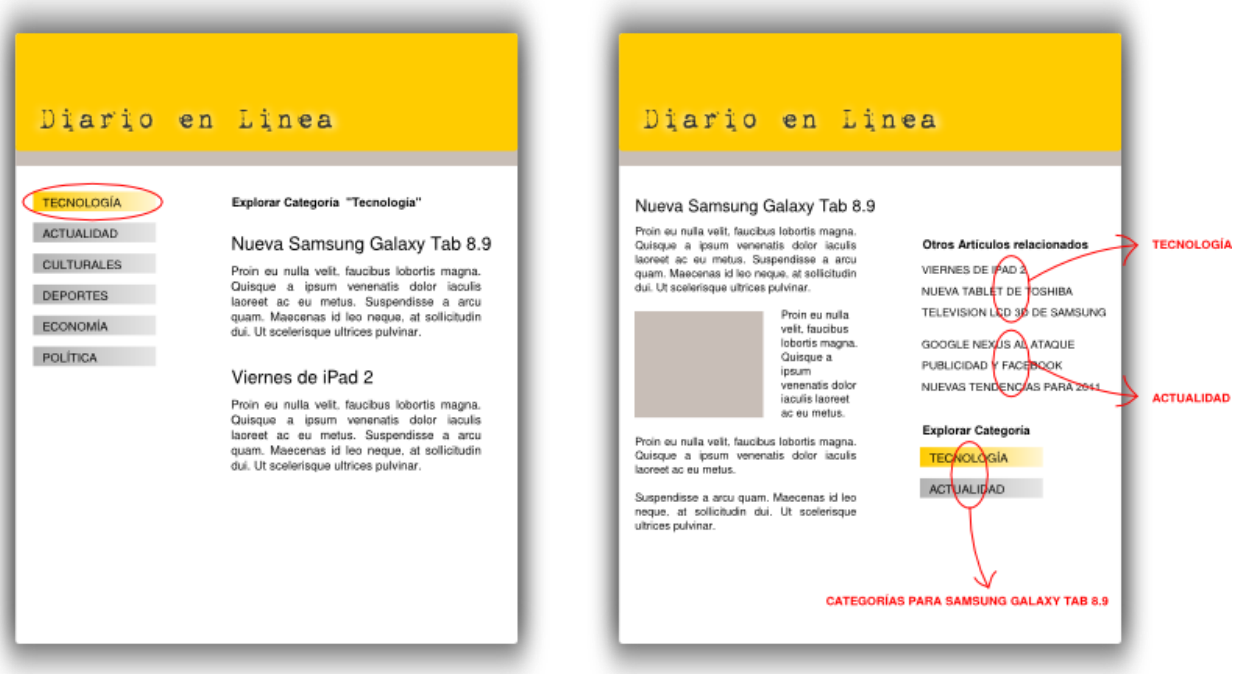


Ilustración 36. Enriquecimiento por categoría

Búsquedas avanzadas: Si el usuario buscó una noticia por título, sección, contenido, y luego que el sistema devuelve los resultados se accede a una noticia en particular, se debería mostrar un listado con las otras noticias que estaban incluidas en los resultados de la búsqueda efectuada. Este listado remplazaría al de noticias relacionadas pues ahora su principal interés es encontrar lo más rápido posible una noticia específica, no explorar una categoría.

Los resultados también se podrían refinar luego de la búsqueda, por lo que las operaciones correspondientes deberían estar disponibles y a la vista del usuario. De igual manera se podría sugerir las palabras clave basadas en búsquedas anteriores, todo esto para enriquecer la experiencia del usuario y ayudarle encontrar lo que busca en el menor tiempo posible.

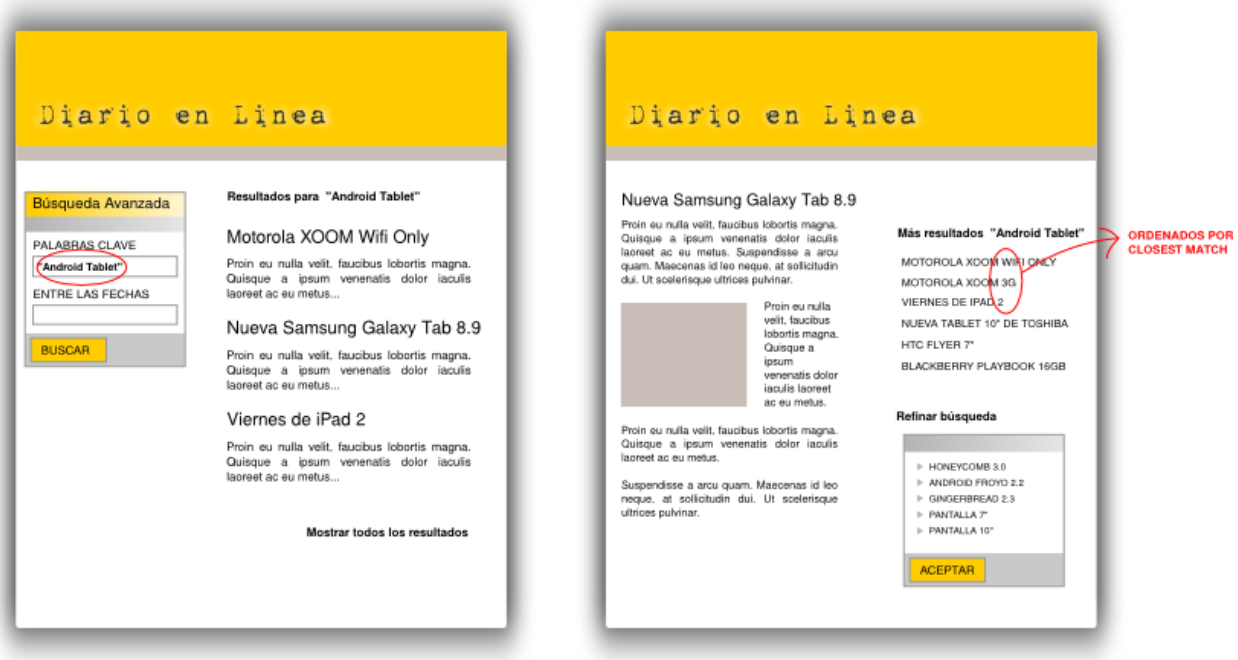


Ilustración 37. Enriquecimiento por búsqueda

Suscripciones: En el caso de que el sistema ofrezca un servicio de suscripción de noticias para ser enviadas por mail diariamente (newletter), podría entonces existir también un listado de las secciones, intereses, entre otros a los que el usuario se encuentre suscrito y que podría ser presentado o modificado desde alguna página siempre que el usuario haya iniciado sesión. Estas operaciones y otras disponibles para usuarios registrados deberían ser fácilmente localizadas en las páginas correspondientes.

Historial: De la misma manera que se presentaban las noticias relacionadas por categoría o búsqueda, puede existir un mecanismo para sugerir al usuario y presentar junto a la noticia principal, aquellas noticias que también puedan ser de su interés en base a su historial de navegación pasada. Una manera de acceder a este concern

podría ser desde un email que llega al usuario con enlaces a noticias recomendadas que acaben de suceder.

5.3 Wikis

Una evolución de los CMSs tradicionales son las wikis. Estos sistemas permiten que varios usuarios en colaboración puedan crear y editar los contenidos de un sitio Web de una manera mucho más sencilla y amigable que sus predecesores, ya que las secciones públicas y de administración no se encuentran claramente separadas, es decir que se tiene la impresión de nunca dejar la página al momento de editarla.

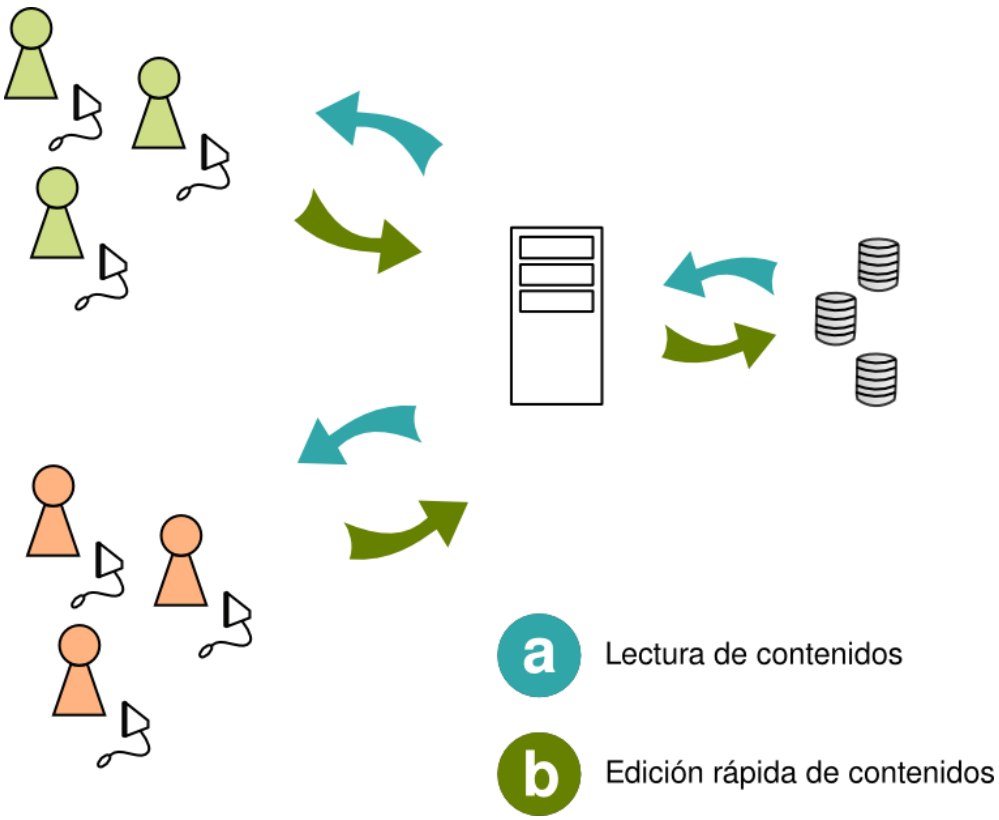


Ilustración 38. Funcionamiento de una Wiki

Las personas que agregan el contenido no son los propietarios o programadores del sitio, sino los mismos usuarios que lo visitan para realizar las lecturas. Además los

contenidos se encuentran vinculados entre sí desde un inicio, ya que es un requisito al crearlos. Estos aspectos han dado lugar a la rápida construcción de enciclopedias en línea, foros de opinión y otras fuentes de información, que normalmente hubieran tomado mucho tiempo en realizarse si solo un equipo pequeño de personas cargara los contenidos con una herramienta más complicada.

Otras características típicas de una wiki es: el empleo de Urls limpias normalmente formadas por el mismo título de la página, buscadores por tags o fechas debido a que guardan un historial de las páginas, capacidad de multi idiomas y la fácil inclusión de enlaces en medio del contenido, por nombrar algunas.

Algunos ejemplos de motores para generar wikis son:

- MediaWiki (usadoporwikipedia)
- DokuWiki
- PmWiki
- TWiki
- PhpWiki

5.4 Agregando CSN a una wiki

Antes de conocer cómo enriquecer una wiki con CSN, veamos cómo se estructura un típico esquema de navegación en estos sistemas, tomando como ejemplo a la Wikipedia.

Al entrar a una wiki observamos un menú de navegación "estático" con enlaces a las páginas más visitadas de la comunidad o a páginas aleatorias. Además se pueden notar las siguientes características:

- Un buscador simple por tags
- Un historial de cambios del artículo que se lee actualmente
- posibilidad de buscar la página equivalente en otro idioma

- herramientas para imprimir / exportar el artículo

Dentro del artículo podemos encontrar los siguientes vínculos:

- atajos a subtítulos dentro del mismo artículo
- enlaces a otros artículos relacionados dentro de la misma wiki
- enlaces a páginas relacionadas fuera de la wiki
- enlaces a archivos descargables dentro de la Wiki
- enlaces a archivos descargables permitidos desde otros lugares
- enlaces a categorías del artículo actual, que llevan a un índice de páginas y subcategorías dentro de cada categoría

Ahora veamos cómo se puede mejorar lo anterior mediante CSN,

Para iniciar la navegación podrían haber ciertos criterios, que bien podrían ser los concerns como: categorías, páginas por país, destacadas, más vistas.

El menú principal puede convertirse en contextual para mostrar enlaces refinados, en base al criterio antes escogido únicamente.

El bloque de páginas relacionadas dentro de la Wiki, debería contener primero aquellas páginas que se enlazan con la actual por medio del criterio escogido. Ya que se infiere que el usuario no necesita todas las páginas que se relacionen con la actual de cualquier manera (tags por ejemplo), sino aquellas que son de su interés, es decir las del concern seleccionado o actual.

Debería darse la oportunidad de cambiar de concern reiniciando la navegación, eso implica un bloque con enlaces que al seguirse personalizan las páginas siguientes de acuerdo al contexto.

El idioma en que se presentará la información podría ser inferido a partir de selecciones previas del usuario. A continuación una ilustración de cómo sería el enriquecimiento.

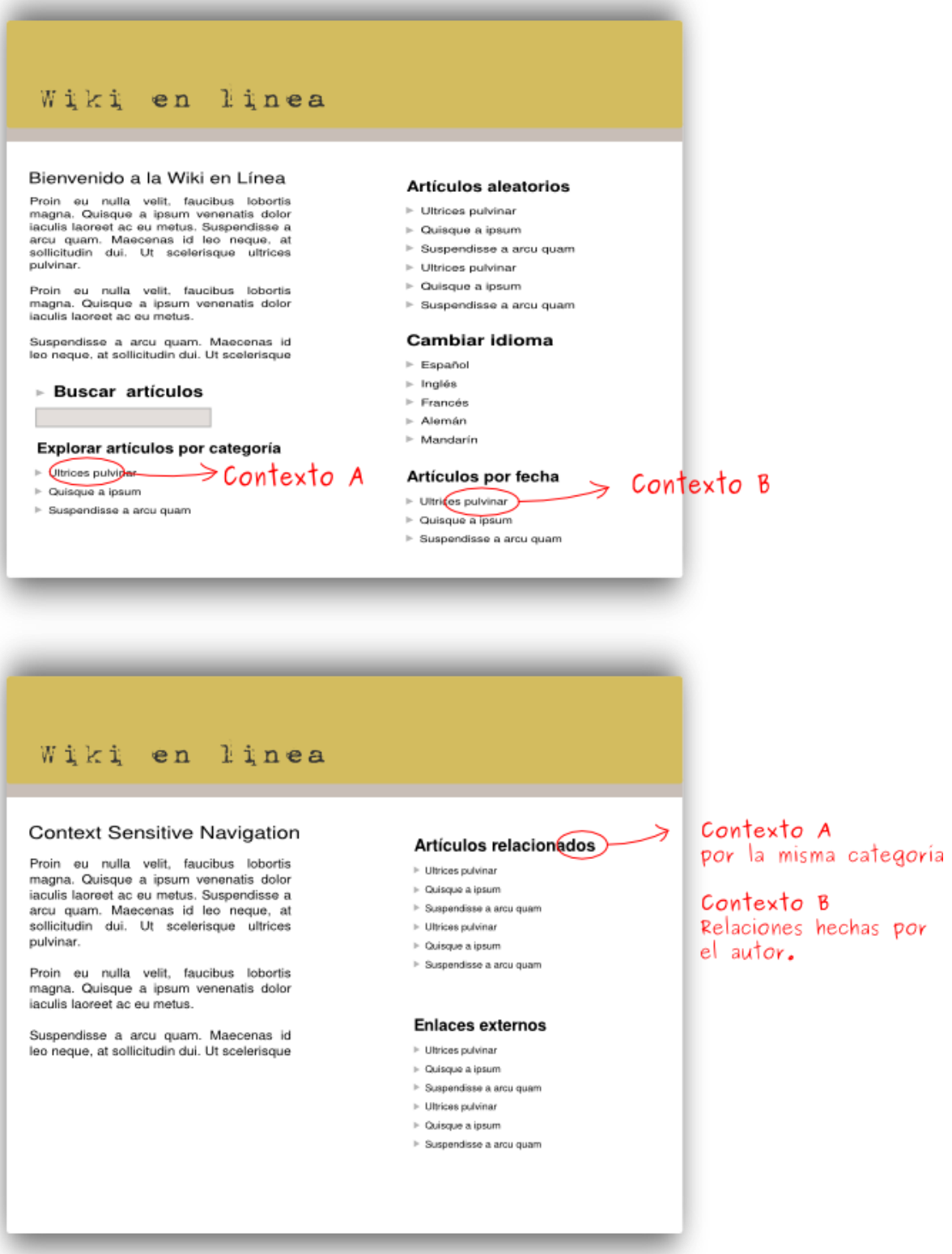


Ilustración 39. Enriquecimiento a una Wiki

Tomemos en cuenta que un artículo podría ser accedido mediante la combinación de dos o más concerns complementarios como: una categoría + un idioma + un historial de navegación, por lo tanto con una arquitectura más elaborada se podría personalizar la interfaz en formas más sofisticadas.

Existe otro contexto para leer un artículo si consideramos el acceso directo al mismo desde un buscador como Google, en este caso deberíamos definir que artículos relacionados se deberían mostrar al no existir un historial de navegación anterior en el sitio Web.

5.5 Blogs

Otra de las aplicaciones web 2.0 que se usan ampliamente en la actualidad son los blogs o web logs. Este término proviene de Web y log que en español significa algo así como un diario o bitácora en la web.

Como su nombre lo indica son sitios de estructura simple que se emplean por su autor para escribir artículos sobre su vida personal, o sobre un tema en particular; y donde los lectores que entran pueden dejar sus comentarios y esperar sus respuestas de parte del autor del artículo.

Los blogs han ido cambiando para contener artículos sobre cualquier tema, y no necesariamente sobre la vida de su autor; es así que podemos encontrar blogs de tipo tecnológico, empresarial, sobre tutoriales, deportivo, educativo, político, institucional, sobre personas famosas, comunitarios, etc.

Ilustración 40. Blog de autor

Pueden contener buscadores, enlaces a otros blogs o artículos, un archivo o historial de artículos por fechas específicas, contenidos multimedia, y casi cualquier cosa que el autor desee publicar.

También entre blogs puede variar la manera en que se manejan los comentarios y se muestran los artículos, asemejándose más a un sitio generado con un CMS que a un blog. Se puede navegar por los diferentes artículos mediante tags, categorías, fecha, etc. Puede permitirse el ingreso o no de comentarios para ciertos o todos los artículos, o requerirse que un usuario se registre para permitirle dejar un comentario.

Existen también herramientas gratuitas para generar blogs, como:

- Blogger
- TypePad Basic
- Blogcloud
- TypePad Plus

- Geeklog
- Drupal
- Blogware
- WordPress

Como se observa algunas de las herramientas también caen dentro del ámbito de los CMSs, ya que por ejemplo en el caso de Wordpress éste evolucionó de un blog a un gestor de contenidos para sitios de todo tipo.

No vamos a dar detalle sobre la forma de enriquecer un blog con CSN, debido a que sería muy parecido a los dos casos anteriores para los CMSs y las Wikis. Recordemos que a fin de cuentas un blog sería una página web de autor.

5.6 Resumen

Como se ve existen muchas alternativas para enriquecer una aplicación web mediante CSN, y este campo aún no ha sido ampliamente explotado, lo que se evidencia en los simples esquemas de navegación que se encontraron al analizar los blogs, sitios generados con CMSs, wikis, shopping carts, etc.

Al separar una aplicación en diferentes concerns en una etapa temprana en el desarrollo de software, también estamos separándola en diferentes esquemas de navegación; los cuales ayudan al usuario a encontrar lo que busca más eficazmente, puesto que los enlaces sugeridos depende de las acciones del usuario, y no del criterio del programador o del diseñador del sitio.

Más adelante demostraremos que es posible crear sitios con CSN de manera sencilla, ya que el diseño de la arquitectura es más simple de lo que parece. Sin embargo el verdadero trabajo está en definir lo que debería aparecer en cada esquema de navegación, dependiendo del uso que se le dé a la aplicación web y de la audiencia de usuario que visitan el sitio.

6. Diseñando un Framework que soporte CSN

6.1 Introducción

Nuestro objetivo será el de construir una estructura básica que soporte CSN y que permita el desarrollo rápido de aplicaciones de este tipo. Nuestro preocupación en este capítulo será el de adaptar esta estructura para soportar los diferentes cambios de contexto, mas no el cómo podemos obtener la información del mismo.

Así mismo existen varias alternativas para su implementación con el fin de permitir construir una arquitectura desde su inicio, o de acoplarlo como componente externo a un framework ya existente.

Para qué un Framework?

Debemos primero definir qué entendemos por framework:

Un framework para aplicaciones web es un software o conjunto de librerías, que está diseñado para dar soporte al desarrollo de sitios y en general a la construcción de cualquier aplicación web. Entonces un framework trata de facilitar aquellas actividades comunes realizadas durante el desarrollo de la aplicación, como por ejemplo: acceso a la base de datos, uso de plantillas, manejo de sesiones, separación de aspectos de programación; además de promover la re utilización de código.

Muchos frameworks basan su estructura principal en el patrón de arquitectura conocido como MVC con el fin de separar el modelo de datos, las entradas del usuario y la interfaz gráfica en 3 componentes que puedan evolucionar de manera independiente. Vamos a tomar como punto de partida para el framework esta arquitectura ya que es una metodología bien probada y utilizada ampliamente. Es orientada a objetos lo que promueve la re utilización de código. Además al haber sido adoptada por otros frameworks nos resultará más sencillo incluir nuestro módulo de CSN en la mayoría de ellos.

Otra ventaja de trabajar con componentes separados, es la posibilidad de extender el framework por medio de la adhesión de módulos, y de esta manera proveer nuevas funcionalidades asociadas con las aplicaciones Web y promover la participación de nuevos programadores en nuestro trabajo

Algunas de las características comunes de un framework para aplicaciones web y que deseamos incluir en el nuestro son:

Seguridad: Permite identificar los usuarios de la aplicación, y restringir el acceso a funciones basadas en algún criterio definido o mediante listas de control de acceso. Nuestro aporte a este aspecto será de agregar permisos a objetos cuando se acceden desde un concern específico.

```
$this->Acl->role = "Usuario";
$this->Acl->concern = "Category";
$this->Acl->allow("Page.view");
```

Acceso a Base de Datos: Permite acceder a varios manejadores de bases de datos sin realizar cambios en el código; en ambientes orientados a objetos realizar el mapeo automático de objetos a registros mediante ORM (mapeadores objeto – relacional); tener herramientas para la fácil migración de datos y soporte transaccional. Nuestro aporte será agrupar las consultas según el contexto de uso en clases diferentes, de esta manera permitimos que un desarrollador se concentre en un solo concern de forma independiente.

Mapeo de URLs: A través de expresiones regulares tener un mecanismo para traducir URLs permitiendo una lectura más "amigable", y la mejor indexación de páginas en motores de búsqueda. Se podrían utilizar prefijos para denotar el concern activo, de esta manera si accedemos a una misma página desde concerns diferentes la URL se verá distinta, haciendo más entendible la navegación.

Un ejemplo sería el siguiente:

- Dominio/accesorios-teléfonos/cargador-5Watts
- Dominio/artículos-destacados/cargador-5Watts

Utilización de Plantillas: Aplicando plantillas o temas y el uso de variables para aquellas partes dinámicas donde se insertan los datos extraídos de una base de datos, se puede reducir dramáticamente el número de páginas en un sitio. Además con el uso de etiquetas especiales para las variables, se puede automatizar la actualización de ciertas zonas y reducir la programación. Nuestro aporte a este punto sería el de cambiar las plantillas al cambiar de contexto, así cuando el usuario acceda a una página desde un concern específico el look &feel del sitio será distinto.

Caché : Permitiendo tener una copia de documentos web a la mano con el propósito de reducir el uso del ancho de banda, y peticiones al servidor.

Ajax: Permite realizar intercambios de menor información con el servidor de forma asíncrona, de manera que no se tiene que recargar la página web cada vez que el usuario pide un cambio. Esto aumenta la interactividad con la página, la velocidad y la usabilidad.

Navegación Sensible a Concerns: Ese será nuestro aporte a los frameworks web comerciales. La idea es permitir personalizar la información, y la manera en que ésta se muestra dentro de una página web, dependiendo del interés específico del usuario en un momento determinado. Este aspecto se mezcla con los antes mencionados para lograr su objetivo.

Otras características que se pueden encontrar:

Creación y consumo de servicios web, configuración automática de ciertos parámetros,

paginación, generación de HTML para ciertos componentes gráficos, etc.

6.2 Principales Bloques a considerar

Nuestro objetivo será el de construir una estructura básica que soporte CSN y que permita el desarrollo rápido de aplicaciones de este tipo. Nuestra preocupación en este capítulo será el de adaptar esta estructura para soportar los diferentes cambios de contexto, mas no el saber cómo podemos obtener la información del mismo.

Para utilizar CSN primero debemos explicar primero un patrón ampliamente utilizado en programación en la Web llamado Model-View-Controller o conocido simplemente como MVC por sus siglas en inglés. Luego lo extenderemos para que soporte CSN y finalmente demostraremos su aplicación en un gestor de contenidos simple.

El patrón de arquitectura Modelo – Vista – Controlador

MVC es una metodología de desarrollo para separar los aspectos relacionados con los datos, la presentación y las entradas de usuario en componentes especializados. Su principal objetivo es el de proveer a los usuarios de una interfaz para manipular múltiples vistas de datos como si trabajaran con objetos del mundo real. Su primera implementación se puede encontrar como parte de la librería de clases de Smalltalk para el dispositivo PARC de Xerox.

Componentes

El modelo se refiere a los datos y funcionalidad de negocio de la aplicación. Esto es a menudo representado por un modelo del dominio donde los objetos buscan modelar entidades del mundo real por medio de la representación de sus propiedades y su comportamiento.

La **vista** es la representación visual del modelo y está compuesta de las pantallas usadas dentro de la aplicación.

Finalmente el **controlador** es un componente que corresponde a los eventos del usuario tales como entradas de datos y comandos iniciados desde el teclado o ratón. Su responsabilidad es la de actuar como un puente entre el usuario humano y la aplicación, permitiendo al usuario interactuar con la pantalla y los datos. Solamente un controlador puede actuar a la vez para tener el control sobre los eventos producidos en la aplicación.

Colaboraciones entre Clases

Dentro del patrón MVC, una triada modelo, vista y controlador existe para cada objeto creado para ser manipulado por el usuario.

El modelo representa el estado, estructura y comportamiento de los datos siendo vistos y manipulados por el usuario. Éste no contiene un enlace directo con la vista o el controlador, y puede ser modificado por ellos u otros objetos del sistema. Cuando se requiere una notificación a la vista para que ésta se actualice, el modelo emplea el patrón **Observer** para enviar los mensajes informando a sus dependientes que los datos han cambiado.

La vista y el controlador trabajan juntos para permitir que el usuario pueda ver e interactuar con el modelo. Cada vista está asociada a un solo controlador y viceversa. Ambos componentes mantiene una relación directa con el modelo.

Para entender mejor la manera en que el modelo notifica de los cambios a la vista, se recomienda buscar información adicional acerca del patrón de diseño Observer. A continuación se presenta un diagrama de clases explicando la colaboración entre los componentes de la triada MVC, tomando en cuenta el patrón Observer. **¡Error! No se encuentra el origen de la referencia.**

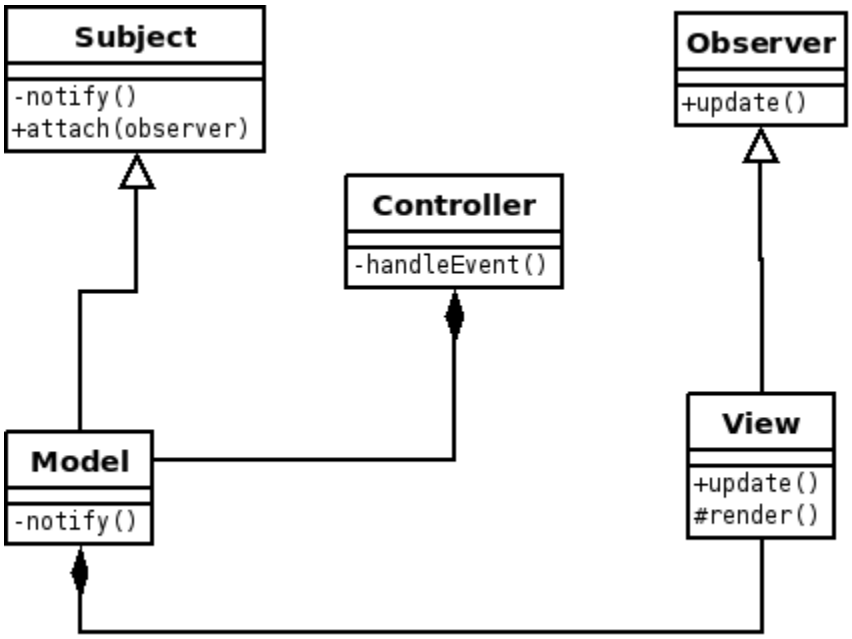


Figura 1. Patrón MVC

Una suposición común y equivocación acerca de la relación entre los componentes del MVC es el objetivo principal del Controlador. Normalmente se cree que su papel es el de separar y comunicar al modelo y a la vista, pero aún cuando ciertamente MVC desacopla la capa de negocios de la aplicación de la de presentación, esto se alcanza por la aplicación del patrón Observer más no a través del Controlador. El trabajo del Controlador es el de mediar entre el usuario y la aplicación, no entre la vista y el modelo.

Actualmente el patrón MVC no se encuentra implementado tal como se lo definió en un inicio. Éste ha cambiado para irse adaptando a las aplicaciones y al contexto de uso. Es así que al contrario de lo que afirmábamos en el párrafo anterior acerca del verdadero rol del Controlador en la triada, existe una manera más pasiva de de la responsabilidad del modelo sin hacer uso del patrón Observer.

En este caso es el controlador el que se responsabiliza de comunicar a la vista acerca de los cambios en el modelo y de ordenar la correspondiente actualización de la presentación en pantalla, sin embargo en nuestro proyecto usaremos el patrón de la manera tradicional.

6.3 Alternativas para implementar CSN

Recordemos la definición que dimos de concern en capítulos anteriores,la cual decía que es cualquier asunto de interés o atención en un programa. Comúnmente se los utiliza como sinónimos de características o comportamientos de una aplicación.

Diversos trabajos se han realizado en el campo de la definición de aspectos en la etapa de requerimientos, con el propósito de identificar y separar tempranamente el código en módulos transversales fácilmente manejables y que puedan ser implementados de manera independiente. Algunos de estos esfuerzos ya fueron mencionados en el capítulo correspondiente a la separación de concerns.

Los aspectos a nivel de requerimientos, presentan a los stakeholdersconcerns que cortan transversalmente al dominio del problema, teniendo el potencial de impactar ampliamente en términos de alcance, priorización y diseño arquitectónico.

Existen diversas formas de poder separar concerns en componentes diferentes. En nuestro primer intento de implementación de CSN, se toma en cuenta la programación orientada a objetos y los patrones de diseño.

6.3.1 Integración con el Patrón State

Para representar los diferentes aspectos en nuestro sistema, vamos a utilizar el patrón State como una manera limpia de cambiar el tipo de la instancia que representa a la vista que toma el control en un momento dado de la ejecución. De esta manera dos actores entran en juego: el Modelo y el Estado, donde el primero contiene una instancia

del segundo y puede intercambiarlo con cualquiera de las opciones disponibles según sus necesidades mientras el programa continúa su ejecución.

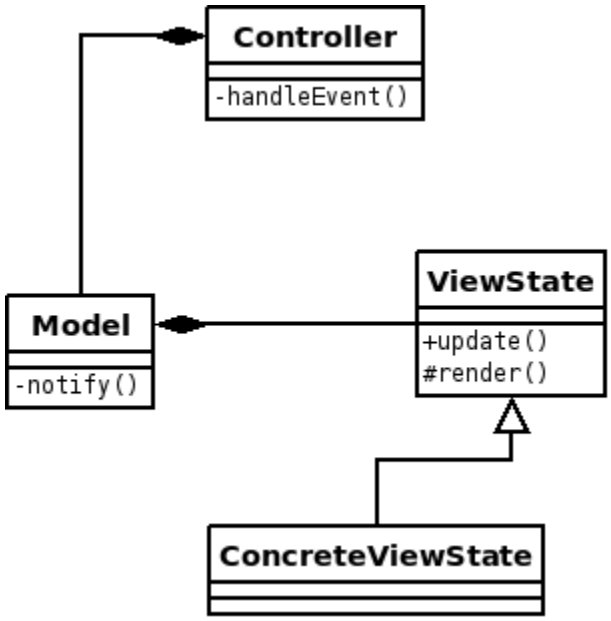


Figura2.Patrón MVC + State

Volviendo a nuestra estructura MVC, esta se puede modificar para que de esta manera se pueda seleccionar una vista específica en tiempo de ejecución, dependiendo del estado actual del contexto, o lo que da lo mismo el concern de interés actual del usuario como se muestra en la Figura2. Resulta conveniente que cada vista concreta hereda de una misma superclase para redefinir los métodos únicamente cuando sea necesario, puesto que no es obligatorio modificar toda la presentación ante un cambio de contexto.

Si partimos de cualquier framework comercial MVC disponible, estos mismos componentes extras se pueden incluir, modificando ligeramente el núcleo del mismo, interceptando el flujo normal antes de enviar las notificaciones a la vista y reemplazándola por el estado correspondiente al contexto actual. Esto implica la

alteración de la arquitectura por lo que resulta una alternativa más conveniente ante la construcción de un Framework desde su inicio, a menos que identifiquemos el punto exacto donde debemos incluir nuestro código. Sin embargo no es recomendable modificar código bien probado y a veces podría haber problemas de licenciamiento.

Usando técnicas y patrones bien conocidos, hemos creado una estructura básica con soporte para navegación sensible a concerns, sin embargo aún estamos lejos de lograr nuestro objetivo pues debemos considerar todos los posibles lugares donde se requiera agregar CSN. El flujo de los acontecimientos quedaría de esta manera.

- El programa principal recibe la orden de manipular un evento desde un controlador concreto, por ejemplo: la carga de una página web.
- Desde allí se crea una instancia del controlador involucrado y se hace un llamado al método correspondiente para el evento requerido. Para realizar este paso usaremos una herramienta poderosa como lo es *reflection* para crear el código necesario en tiempo de ejecución.
- Suscribimos la vista correspondiente al modelo según el concern actual, de manera que se le informe de los cambios sobre el modelo observable. La forma de conocer qué concern se debe usar requiere un mapeo con los controladores y eventos, algo muy parecido al uso de listas de control de acceso.
- El siguiente paso es manipular el evento desde el controlador, y entonces modificar el modelo que representa la página en pantalla, para luego comunicar de estos cambios a la vista activa mediante el patrón *observer*. Esta vista se encarga de realizar los cambios pertinentes en pantalla.
- Las modificaciones en cada vista se basan en los enriquecimientos vistos en el capítulo referente a CSN, lo cual incluye nuevas operaciones, nuevos enlaces y/o contenidos.

Esta colaboración puede entenderse mejor mediante un diagrama de secuencia, como se muestra en la Figura 3.

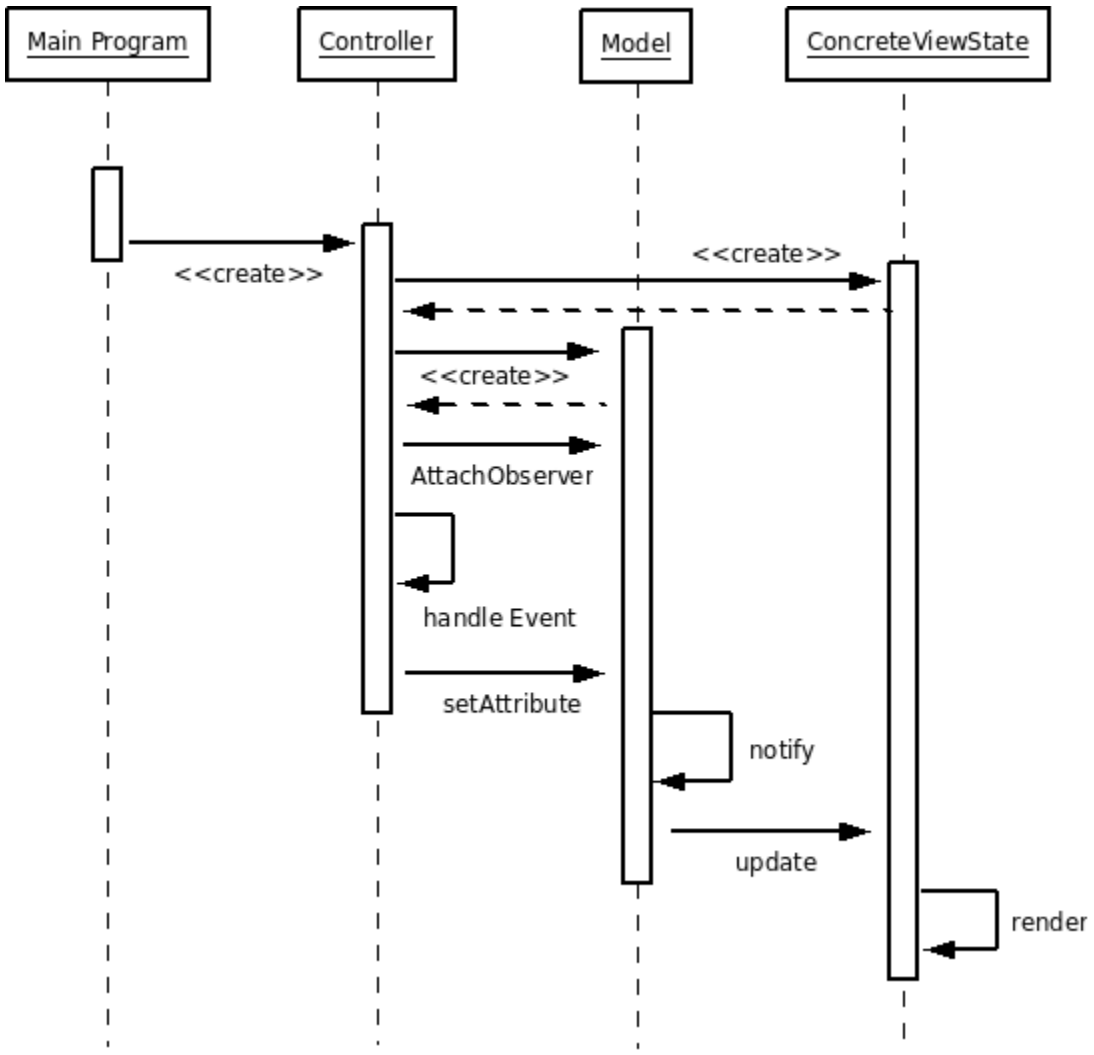


Figura 3. Diagrama de secuencia para la Integración con el Patrón State

Ahora veamos un ejemplo de este proceso desde la clase abstracta del controlador ante la captura de un evento. Aquí intervienen todos los componentes antes mencionados en el orden descrito en el diagrama de secuencia. Todo nuevo controlador heredará este comportamiento. El código se encuentra en lenguaje PHP.

```
class Controller{
        public function __construct($event){
                $this->model = new $modelName;
                $this->model->attachObserver(new $currentStateView);
                $this->handleEvent($event);
                $this->model->update();
        }
}
```

En el código anterior se observa como en el constructor de la clase abstracta controlador, se instancia el modelo correspondiente utilizando reflection para luego atar a éste la vista implementada para el concern dominante en ese momento. A continuación se manipula el evento (Ejemplo: mostrar página) lo que llamará al método apropiado en el controlador concreto para alterar el modelo, finalmente se notifica de los cambios al observador lo que actualizará la vista en pantalla, enriqueciendo la experiencia del usuario para el concern actual.

Notificando los cambios de concern

Si bien para cada concern hemos creado una vista específica que se encargará de mostrar la información adecuada en el momento adecuado, necesitamos sugerir la manera de enviar las notificaciones de cambio de concern que ocurren durante toda la ejecución y que se encargarán de reemplazar estas vistas en tiempo de ejecución.

Una manera de hacerlo sería mapeando los cambios de concern junto con los controladores y eventos que disparan estos cambios en un matriz, tal como se muestra en la Ilustración 41, lo que resultaría en algo parecido a una lista de control de acceso que sería consultada al momento de manipular el evento desde el constructor del controlador.

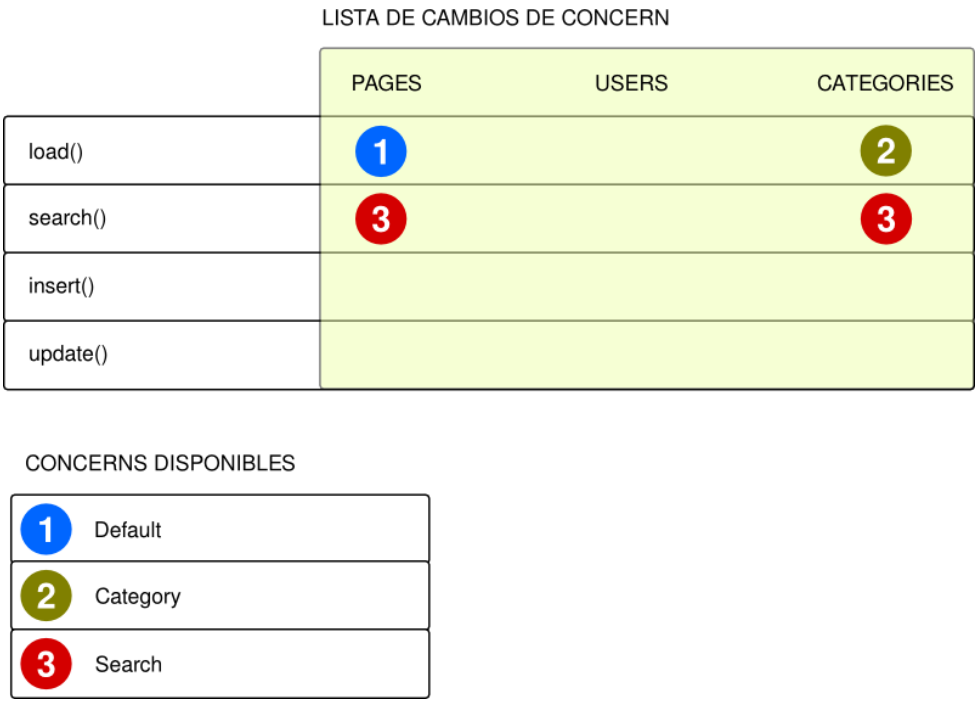


Ilustración 41. Lista de control de cambios de Concern

El código de la sección anterior se modificaría de la siguiente manera para incluir la vista correspondiente al nuevo concern:

```
class Controller{
        public function __construct($event){
                $this->model = new $modelName;
                $nextStateView= $this->Ccl->getConcern($event);
                $this->model->attachObserver(new $nextStateView);
                $this->handleEvent($event);
                $this->model->update();
        }
}
```

La línea resaltada en amarillo muestra la manera de consultar a esta lista de control de cambios de Concern (CCL) presentada como un atributo de la clase controlador. Aquí se envía el nombre del evento al método getConcern() que retorna el nombre de la vista que deberá ser instanciada y agregada al modelo como su observador. En el caso de encontrarse en un elemento en la matriz del CCL se cargaría la vista por defecto de la cual todas las demás heredan su comportamiento. Se deber tener en cuenta que la clase para la vista solicitada debe estar implementada caso contrario se producirá un error en tiempo de ejecución.

En una sección posterior explicaremos como agregar otros aspectos como seguridad, manejo de plantillas, etc. Por el momento veamos una alternativa para la implementación de CSN que permite notificar los cambios de concern de una forma menos intrusiva.

6.3.2 Integrando CSN mediante AOP

Otra forma de notificar los cambios de concern pero de una manera más transparente se consigue usando AOP.

Para lograr esto es necesario crear un aspecto para cada concern de navegación identificado en el análisis de la aplicación. En esta alternativa no es necesario tener varias implementaciones para la vista de la triada MVC, sino que tejemos el código que contiene los enriquecimientos específicos para un concern a una única vista justo antes de mostrarse en pantalla al momento en tiempo de compilación.

De esta manera al provocarse un evento en particular, el método de la vista encargado de mostrar los cambios ahora tendrá también la implementación para las nuevas operaciones, contenidos o enlaces; justo antes o después del código original que no se puede modificar.

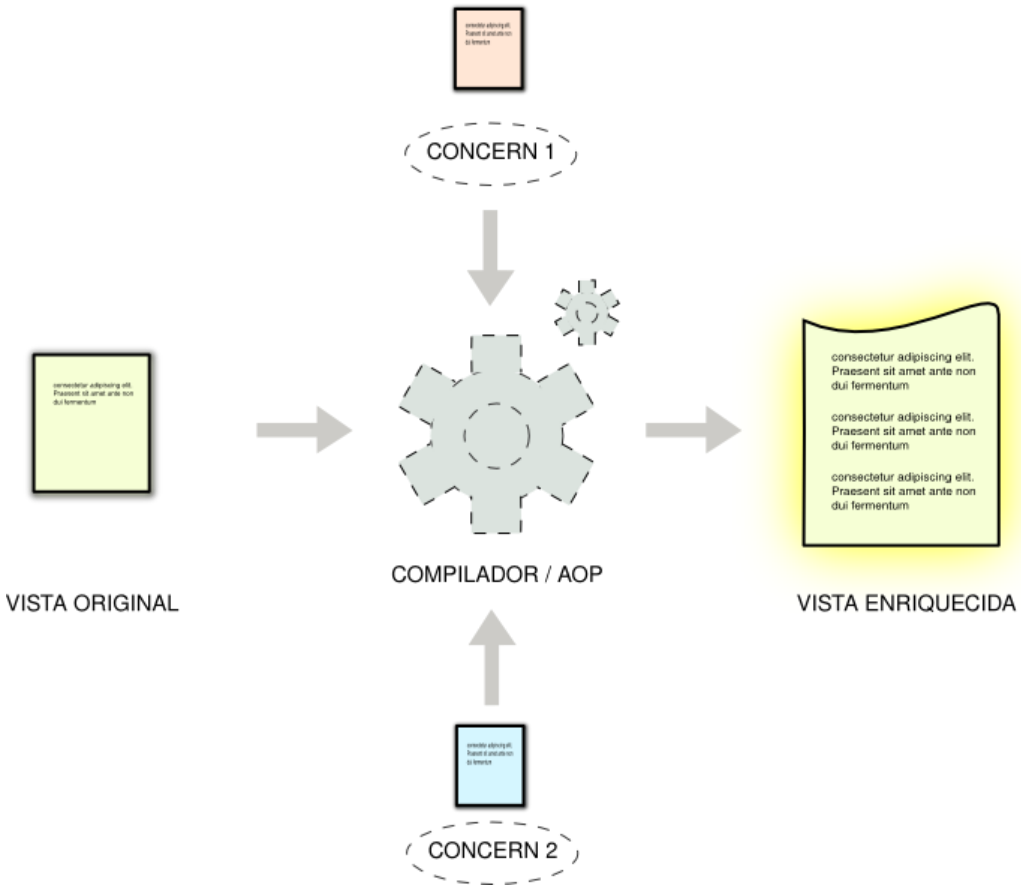


Ilustración 42. Enriquecimiento mediante AOP

Este nuevo código es diferente para cada concern, por lo que se debe implementar el mecanismo que seleccione el aspecto a utilizar para el contexto actual, pudiendo ser nuevamente una lista de control de cambios de concern como en la alternativa anterior, o aplicando nuevamente AOP como se explica a continuación.

Se evidencia que esta alternativa es más limpia que la anterior, pero tiene sus desventajas como el no permitir re utilización de código a través de herencia, AOP requiere mayores exigencias para su implementación y la construcción de los aspectos no es tan sencilla. De igual manera no siempre se podrá incluir AOP en una aplicación o en un framework ya existente, sin tener que modificar en algo la arquitectura original a

menos que ésta ya la soporte desde un inicio. Existen frameworks con soporte nativo para AOP donde esta alternativa sería la más adecuada para la implementación de CSN.

Notificando los cambios de concern con AOP

También usando AOP podemos conseguir un envío de notificaciones no intrusivo creando de igual manera un aspecto para cada concern, e incluyendo el código necesario al controlador para atar la nueva vista antes de la ejecución de un evento. El aspecto consta de los siguientes elementos para mapear los cambios de concern.

- Pointcuts que especifican los métodos en el controlador vigente donde se produce el cambio de concern, ya que estos cambios se dan al producirse un evento o conjunto de eventos en particular.

- Los advices o código a ejecutarse antes de dispararse el o los eventos involucrados. Se ha decidido que sea insertado antes de leerse el código para el método relacionado, de manera que éste método ya se encuentre implementado en la vista correspondiente al nuevo concern.

- El o los joinpoint derivados, que son los puntos dentro de los métodos donde se debería insertar el nuevo código, y que como se explicó en el punto anterior se encuentran justo antes de la primera línea de código de cada método.

Los aspectos se encuentran definidos en un archivo XML, que se tejerá con el código original del controlador en tiempo de compilación. Este archivo tendría la siguiente forma, suponiendo que se ha incorporado un concern para un estado de la aplicación por defecto, otro para la navegación por categorías, y otro para búsquedas.

```
<aspect>
<!--CATEGORY CONCERN-->
<pointcut auto="before" function="showCategory" class="PageController">
<![CDATA[
    $this->model->setConcern('Category');
```

```
   ]]>
</pointcut>

<!--DEFAULT CONCERN-->
<pointcut auto="before" function="showDate" class="PageController">
<![CDATA[
      $this->model->setConcern('Default');
   ]]>
</pointcut>
<!--SEARCH CONCERN-->
<pointcut auto="before" function="search" class="PageController">
<![CDATA[
      $this->model->setConcern('Search');
]]>
</pointcut>
</aspect>
```

El controlador original no conoce nada acerca de CSN y del XML que se utiliza para notificar los cambios de concern, Este XML se teje con el controlador mediante AOP justo antes de ser usado, lo que provoca que antes de cada evento exista la llamada a un método en el modelo, que ordena el cambio de Vista acorde al nuevo concern. Cabe mencionar que esto solo se produce cuando el evento (método relacionado en el controlador) se encuentra registrado en el archivo XML.

Al haber examinado las dos primeras alternativas de implementación de CSN, podríamos deducir que es posible combinar las técnicas para obtener una estructura híbrida que use el patrón State y AOP. Con esto tendríamos la posibilidad de tener varias clases para implementar los comportamientos específicos de cada concern y

notificar los cambios de manera transparente mediante programación por aspectos. De igual manera podríamos tener CCLs almacenadas en una base de datos para las notificaciones y AOP para el enriquecimiento transparente.

Es necesario analizar sus beneficios y desventajas antes de construir nuestro Framework desde el comienzo.

6.3.3 Implementación mediante XSLT

Este método es el que se describe en el trabajo original de CSN, el cual hace uso de transformaciones XSL (XSLT) para renderizar en pantalla los resultados obtenidos de un enriquecimiento por CSN.

La idea central es especificar tanto los nodos como los tipos de roles (enriquecimientos) en archivos XML. Cuando los nodos son poblados con información se tejen los enriquecimientos y se traducen a interfaces finales mediante especificaciones XSLT para obtener el HTML a mostrase en pantalla.

```
<producto>
    <nombre>Guitarra FenderStratocaster Eric Johnson</nombre>
    <precio>$2.499</precio>
    <características>……</características>
</producto>
```

```
<productos-relacionados>
    <producto>
            <nombre>Guitarra Fender Eric Clapton</nombre>
            <costo>$3.100</costo>
        </producto>
        <producto>
            <nombre>Guitarra FenderStratocaster American Standard</nombre>
```

```
            <costo>$1.099</costo>
        </producto>
</productos-relacionados>
```

Aunque la implementación de XSL es simple, se dificulta cuando tenemos un proceso más complicado que necesite de lógica de programación para agregar los enriquecimientos. En esos casos es preferible alternativas orientadas a objetos como en apartados anteriores.

Conclusión.-

Luego de las alternativas dadas se ha optado por seleccionar la primera que se refiere a la utilización del patrón State para implementar los enriquecimientos en clases separadas, con la posibilidad de heredar funcionalidades de un concern por defecto (core concern). En cuanto al método para notificar los cambios de concern se ha escogido las listas de control de cambios de concern pero almacenando esta información en una base de datos en lugar de tener un archivo XML.

Esto permite que un usuario administrador pueda manipular estas listas desde la aplicación, con la modificación de poder seleccionar la URL de una página donde se produzca el cambio, en lugar de una lista de controladores y eventos que dificultarían el entendimiento.

Más adelante damos más detalles de la implementación y plantemos la estructura final del Framework junto con sus componentes extras requeridos para su funcionamiento.

6.4 Acoplando otros componentes

Para que nuestro Framework sea funcional y empecemos a usarlo para construir aplicaciones más complejas, necesitamos agregarle características como las que mencionamos al inicio de este capítulo. Para probar que sí es posible acoplarlas a la

estructura básica, veamos cómo se implementarían las que se mencionan a continuación.

- Seguridad mediante permisos a usuarios y roles
- un mecanismo para tener acceso global a las variables del HttpRequest y de Sesión
- un mecanismo de persistencia
- creación de plantillas HTML (masterpages) sensible a cambios de concerns

Aunque no es nuestro propósito en esta tesis analizar e implementar estas funcionalidades, al menos debemos dejar planteada la arquitectura apropiada, que permitirá extender el framework con estas características. En otras palabras, necesitamos especificar de qué manera y en qué lugar deberían agregarse las partes, para no dañar el diseño de clases que obtuvimos para CSN.

6.4.1 Usuarios y Roles

La forma ampliamente aceptada para la implementación de este aspecto es mediante el uso de listas de control de acceso de la siguiente forma:

Existen dos componentes bien definidos para la asignación de permisos. Los primeros son los objetos a los que el usuario pretende acceder como por ejemplo una página, un controlador, un evento o en nuestro caso un concern. Los segundos son los grupos o roles a los que pertenece el usuario; a estos grupos se les puede otorgar o negar permiso para acceder a los objetos antes mencionados.

Vamos a considerar a los usuarios que no han iniciado sesión en el sistema como miembros de un rol denominado "anónimos".Entonces al producirse un evento y justo antes de llamar al método correspondiente del controlador, validamos si el rol al que el usuario pertenece tiene permisos para utilizar ese método dentro del concern actualmente activo.

La implementación de este mecanismo puede incluir la utilización de un archivo XML que es parseado para conocer si un rol tiene o no permiso de acceso al objeto consultado. Igualmente se podría almacenar esta información en una base de datos y de esta manera darle la posibilidad al administrador del sitio de otorgar o negar permisos mediante una interfaz de usuario. Esta interfaz luciría como una matriz con un aspecto similar al siguiente.

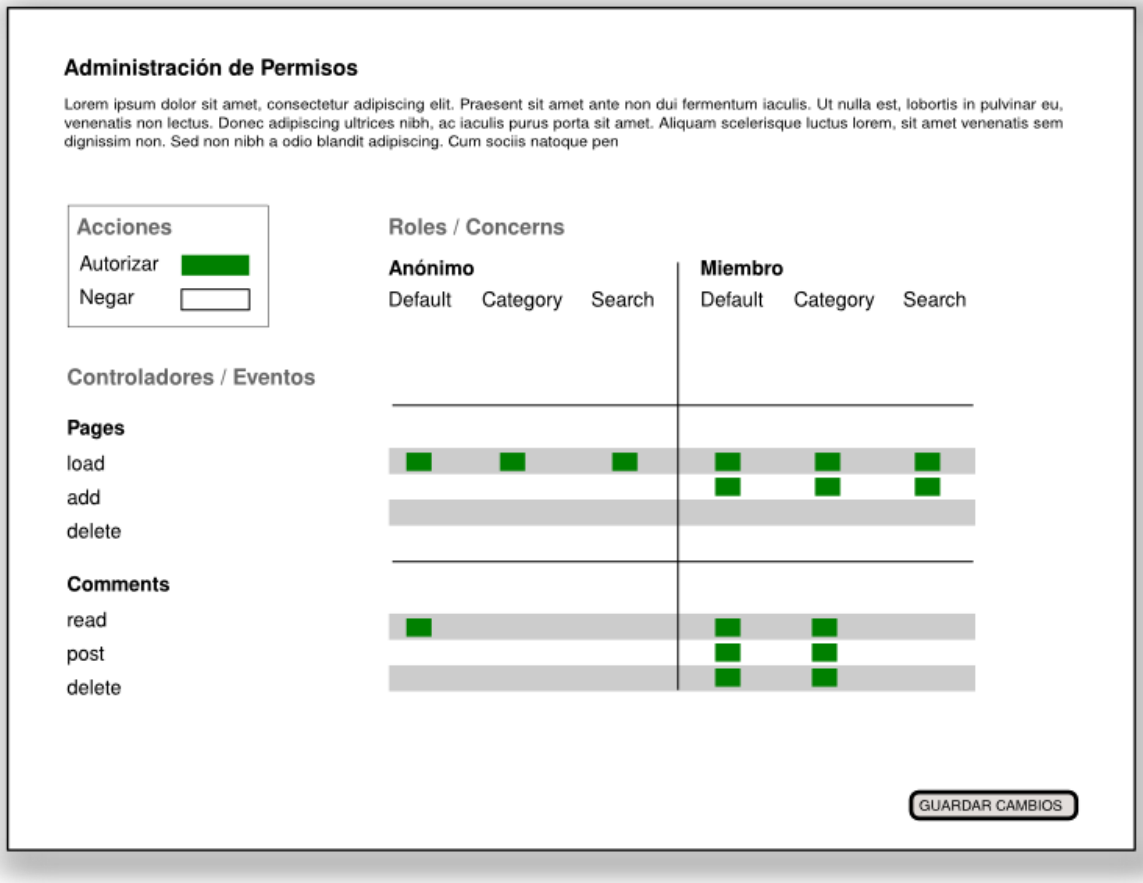


Ilustración 43. Permisos y Concerns

La estructura necesaria para la implementación de este aspecto requiere nuevas clases para los usuarios, roles, las ACLs, recuperar información de la sesión y para los concerns en sí.

Además de los permisos, también se podría configurar en el sistema una página para mostrar errores cuando el usuario no está autorizado, o a donde ir cuando un usuario inicia sesión, o pedirle que ingrese sus credenciales si trata de acceder a un objeto sobre el cual el rol anónimo no tiene permiso.

La colaboración entre ellas se muestra a continuación:

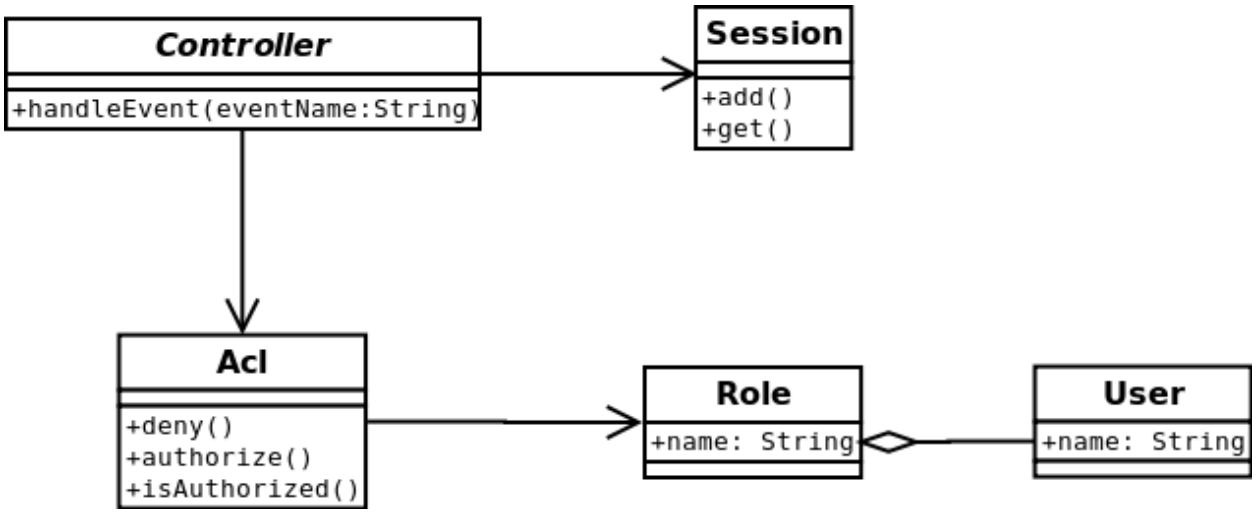


Figura 4. Listas de Control de Acceso para el Framework

El código necesario que se incluye al invocar el método en el controlador desde su constructor se muestra en amarillo.

```
class Controller{
        public function __construct($event){
                $this->model = new $modelName;
                $nextStateView = $this->Ccl->getConcern($event);
                $role = $this->Session->get("currentRole");
                If($this->Acl->isAuthorized($this,$event,$role,$nextStateView)){
                        $this->model->attachObserver(new $nextStateView);
                }else{
```

```
                    $this->redirect("not_authorized");
                }
                $this->handleEvent($event);
                $this->model->update();
        }}
```

En el código anterior podemos apreciar como antes de cambiar de concern o de siquiera manipular el evento solicitado, se consulta en las listas de control de acceso los permisos para el rol del usuario en sesión sobre el controlador, método y concern solicitados. En caso de tener permiso sobre el objeto solicitado se ejecuta el resto de código normalmente, caso contrario el usuario es llevado a una vista predefinida donde se le indicará que no está autorizado para ver esa página.

6.4.2 Acceso a las variables del HttpRequest

Cada vez que se realiza la petición de una página (HttpRequest) a un servidor, ésta va acompañada de algunos parámetros que ayudan al script que reside en el servidor a generar el Html necesario, que se mostrará en el navegador al momento de obtener la respuesta del servidor (HttpResponse).

Estos parámetros viajan de esta manera en la URL (en el caso de usar GET):
http://www.diariolaplata.com.ar/index.php?categoryId=3&orderby=date

En el servidor llegarán dos variables.

- categoryId, con valor "3"
- orderby, con valor "date"

Así sabemos que se pide mostrar la página inicial (index.php), pero también se incluyen las instrucciones para mostrar solo los artículos correspondientes a la categoría con código 3, y ordenadas por fecha.

Es decir, un evento en el controlador activo por ejemplo: PageController, necesitará acceder a estas dos variables para saber que necesita realizar los filtrados necesarios sobre los objetos de la página, y para que finalmente la vista activa refleje estos cambios, generando el HTML necesario que se renderizará en el navegador. Además puede ser que no solo el controlador necesite acceder a estas variables.

Dicho en otras palabras, necesitamos acceder de manera global desde cada componente de la triada MVC o por lo menos desde el controlador, a un objeto que contenga estas variables. Además no necesitamos duplicar instancias de este objeto, para cada clase desde la que accedamos. Esto se puede conseguir fácilmente con la aplicación de los patrones Adapter y Singleton en conjunto.

El diseño tentativo para lograr esto se muestra en la siguiente ilustración.

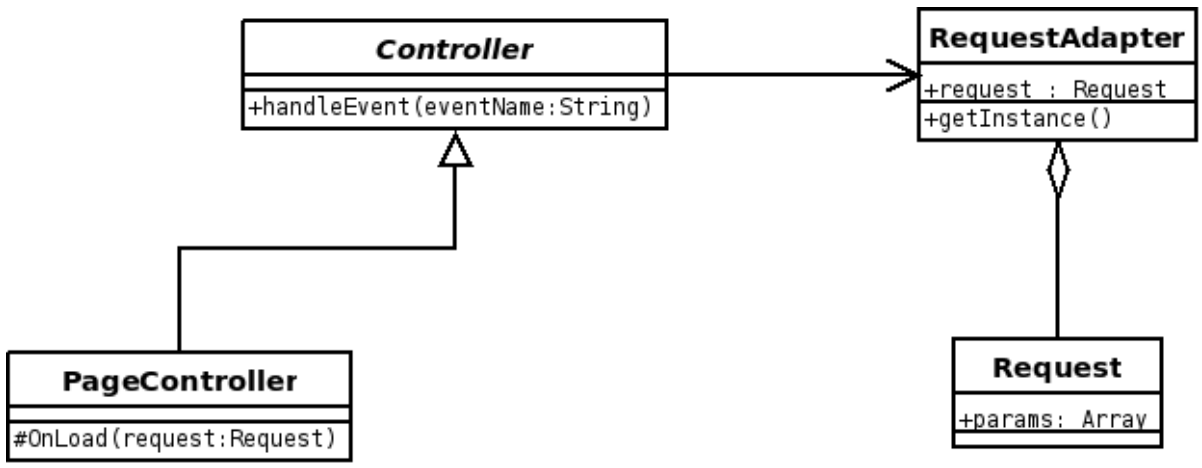


Figura 5 - Acceso a parámetros del Request

En síntesis, se hace una petición al adaptador para que nos devuelva un objeto Request (siendo éste único para toda la aplicación), y a partir de él recuperamos el o los parámetros que necesitemos. Esto también nos permite en caso de ser necesario,

introducir nuevos parámetros y valores al Request mediante este objeto, y permitir que los demás nodos puedan acceder a ellos.

A continuación se muestra un ejemplo de su posible uso, donde el objeto Request es pasado como parámetro a cada método o evento del controlador para su utilización desde la clase abstracta Controller al momento de invocar el evento que se pide

```
public function handleEvent( $eventname ) {

        //invoke local method for $eventname, and pass the request params to it.

                $this->$eventname(RequestAdapter::getInstance());

                $this->model->setState($eventname);

                //notify change of state to observer

}

Ahora dentro de un método del controller concreto.

//shows up a page story

protected function showStory(Request $request){

$this->model->findActiveStory($request->get("storyId"));

}
```

Éste evento permite buscar y mostrar un árticulo de interés, el cual posee un código que es recuperado del Request, mediante el parámetro storyId.

Un acercamiento similar se también puede usar para las variables de Sesión, tal como se sugirió en la sección anterior para consultar el usuario y rol activos.

6.4.3 Persistencia

Ahora bien, nuestro framework no estará completo a menos que especifiquemos una manera de poder guardar nuestros objetos que se encuentran en memoria, al disco duro. Existen varias alternativas para esto:

- Incluir un ORM existente para PHP
- Agregar un data mapper
- Usar nuestro propio mecanismo de persistencia

Cual sea que fuera nuestra estrategia de persistencia, debemos al menos indicar en qué lugar del modelo de clases se debería agregar esta característica. De esta manera probamos que la arquitectura planteada es lo suficientemente escalalable, para permitir extenderla con nuevas funcionalidades cuando sea necesario.

Si bien no requerimos que la alternativa de persistencia pueda ser intercambiada en tiempo de ejecución, ya que generalmente necesitamos utilizar una sola durante todo el tiempo de vida de la aplicación; se necesita que pueda ser configurada antes de empezar un nuevo proyecto. De esta manera si en algún momento deseamos migrar hacia una nueva tecnología de persistencia, solamente haría falta especificar este cambio a través de una constante (E.j. PERSISTENCE_FACTORY), y agregar el nuevo código en clases separadas sin afectar las existentes.

6.4.3.1 Incluyendo el Componente

Vamos a suponer solo como ejemplo que empleamos nuestro propio mecanismo de persistencia. La idea fundamental en este ejemplo es persistir todo el modelo de la triada MVC en el disco duro en forma de un archivo de texto, con toda la estructura de objetos serializada; para luego en un momento determinado recuperar la información

con todos los objetos que serán visualizados (Ej. una instancia de la página con todos sus elementos).

Entonces en una clase a parte tendríamos un código como este:

```
$file = fopen("nombre_archivo","w"); //abierto para escritura

fwrite($file,serialize($model)); //serializar los objetos en memoria y guardar
```

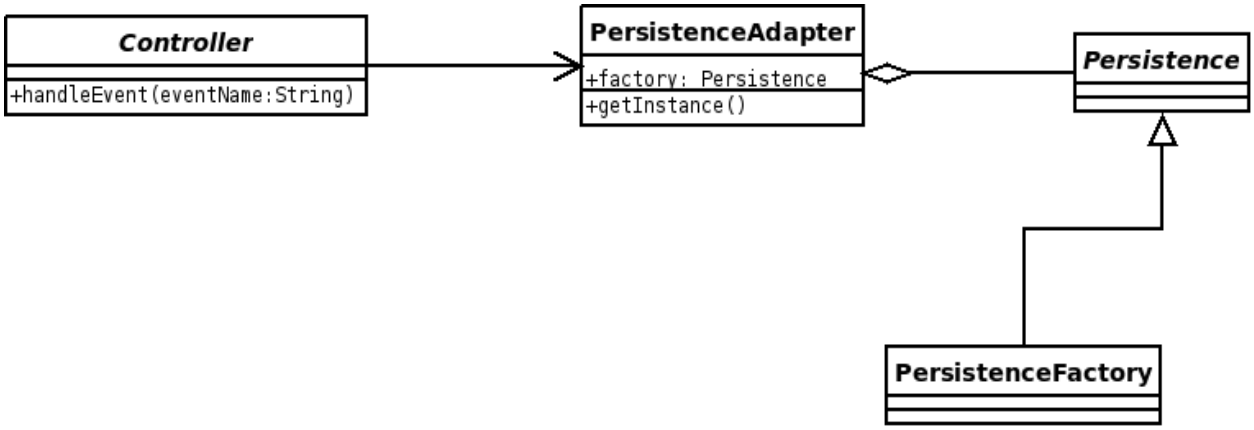


Figura 6 - Agregando capacidades de Persistencia

Luego se para recuperar la información, realizaríamos el proceso inverso en otra función de la misma clase. Finalmente podemos consultar bien sea desde el controlador o desde la vista activa (aunque no sería recomendable), los objetos en memoria, con un código como el que se muestra a continuación.

Tomando en cuenta que sin importar si luego se cambiara la alternativa de persistencia, el código se mantendría siempre igual mientras hagamos una nueva clase por separado con los mismos métodos:

```
$per_strategy = PersistenceAdapter::getInstance();
```

```
$this->model = $per_strategy->Read();
$this->model->getStories();
```

Para lograr esto, nuevamente necesitamos que la instancia de esta "fabrica" de persistencia sea única para toda la aplicación, y que el acceso a ella sea global desde cualquier componente que la necesite; por lo que utilizaremos el patrón Singleton una vez más.

Así mismo si requerimos recuperar el modelo a partir de la base de datos o archivo de texto, tomando en cuenta los pasos necesarios como conectarnos a la base, recuperar información, parsear los objetos serializados, armar la instancia, agregar los sub elementos recursivamente, etc. No importaría realmente que la clase destino sepa cómo se realizó cada paso desde la alternativa de persistencia seleccionada, siempre que devuelva el producto final esperado, por lo que emplearemos el patrón Abstract Factory. De esta manera nuestro mecanismo de persistencia serializando objetos para guardarlos en disco, vendría a ser una fábrica más. Este modelo de clases resultante se parece mucho al acercamiento llevado a cabo por el patrón de persistencia denominado Data Access Object o DAO, ampliamente usado en aplicaciones de JAVA.

Aunque podríamos aplicar esta técnica, resulta poco práctica debido a que desperdiciaríamos gran cantidad de memoria al almacenar toda la estructura. Es así como a continuación se muestra formalmente el diseño de un ORM lo suficientemente apropiado para manejar los aspectos de persistencia en nuestro Framework. Debemos tomar en cuenta que podrían existir muchas otras ideas en trabajos futuros, que en definitiva seguirían los mismos lineamientos señalados hasta entonces para ser implementadas.

6.4.3.2 Trabajando con Bases de Datos

Esta alternativa es mucho mejor que la presentada anteriormente e incluye una capa de abstracción de base de datos, cuya implementación se encuentra en clase y archivos separados de la estructura del núcleo del Framework.

Esta capa funciona como un mapeador objeto relacional, de manera que nos permite olvidarnos de la elaboración de un diseño de modelo de datos y de la escritura de consultas SQL, al dejar que sea este componente quien se encargue de persistir nuestros objetos. Sin embargo, se deben seguir ciertos estándares de nombres de tablas y columnas para que el mapeo funcione apropiadamente, ya que la creación de la base de datos y de su estructura correría por nuestra parte por el momento.

Definiendo los parámetros de base de datos

Los parámetros de configuración a la base de datos se encontrarán en un archivo llamado config.php localizado dentro de la carpeta con el mismo nombre.

```
//persistence strategy

define("PERSISTENCE_FACTORY","Db");

define("PAGE_SIZE",10);

//database configuration

define("DB1_HOST","localhost");

define("DB1_USER","root");

define("DB1_PASSWORD","root");

define("DB1_NAME","tesis");
```

La primera sección titulada "persistencestrategy" contiene dos constantes. PERSISTENCE_FACTORY nos indica el tipo de persistencia que se va a usar, el valor por defecto es Db. Sin embargo, se pueden implementar otras formas de persistencia agregando una clase diferente dentro de la misma carpeta y heredando de PersistenceStartegy; en este caso Db especifica que se va a utilizar la implementación de la clase DbPersistence. La otra constante PAGE_SIZE, sirve para indicar cuantos registros deben ser recuperados a la vez. Esto es útil cuando se necesita paginación, más adelante mostraremos un ejemplo.

La segunda sección "database configuration", contiene los parámetros de conexión en el siguiente orden: host del servidor de base de datos, credenciales: nombre y clave de usuario, y el nombre de la base de datos. Debemos estar seguros que tanto la base de datos como las credenciales estén debidamente creadas y con los permisos necesarios.

Entendiendo el mapeador Objeto - Relacional

La librería necesaria para utilizarlo se carga por defecto antes de llamar a un evento, por lo que no es necesario incluir su importación nuevamente ya que se encuentra disponible en todo momento. Esta importación se puede encontrar en el archivo config, mediante la línea:

```
require_once'lib/plugins/persistence/class.php';
```

Para hacer uso de esta funcionalidad debemos obtener una instancia única y global de la fábrica de persistencia DbPersistence, a través del Adaptador llamado PersistenceAdapter e invocando al método getInstance().

```
$factory = PersistenceAdapter::getInstance();
```

Este código puede ser utilizado en cualquiera de las clases de la triada MVC durante la ejecución de un evento, sin embargo no se recomienda su uso dentro de la clase del modelo, pues la intención de la existencia del mapeador es hacer que el modelo y los datos no necesiten conocerse. El siguiente ejemplo recupera de la base de datos las instancias de Persona que se encuentran en estado activo.

```
$factory = PersistenceAdapter::getInstance();

$personas = $factory->getObjects("Persona" , " estado='activo' " );
```

El uso del mapeador no afecta en ninguna forma al modelo de clases, ya que la implementación de los métodos para guardar los objetos se encuentran en la fábrica de persistencia, es decir en la clase DbPersistence. En otras palabras, una clase del modelo no necesita ser modificada o heredar de otra para almacenarse, sino que ésta es pasada como parámetro de entrada al llamar al método de persistencia correspondiente. El siguiente ejemplo explica lo antes mencionado, en este caso una instancia de la clase persona es guardada en la base de datos.

```
$persona = new Persona();

$persona->setNombre("Gustavo");

$persona->setEdad(40);

$persona->setDNI("98798");



$factory = PersistenceAdapter::getInstance();

$factory->Save($persona);
```

El mapeador se encarga de reconocer la instancia que se le pasa, y guarda la información en la tabla correspondiente. No es necesario definir un identificador único para la instancia o para el registro en la tabla, éste es generado secuencialmente después de ser almacenado. Si luego quisiéramos recuperar la persona simplemente utilizaríamos el siguiente código.

```
$factory = PersistenceAdapter::getInstance();

$persona = $factory->getObject("Persona" , " dni='98798' ");
```

En la siguiente ilustración se muestran el proceso que se lleva a cabo cada vez que utilizamos nuestro mecanismo de persistencia.

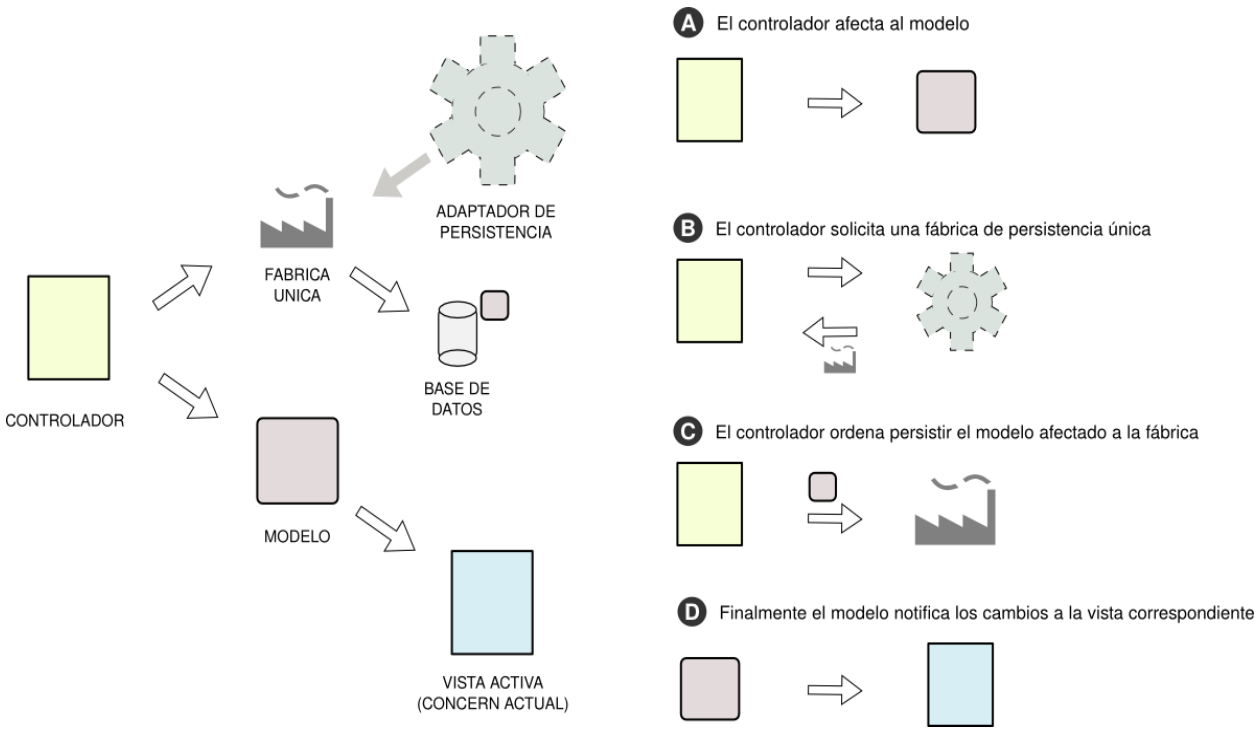


Ilustración 44. Mecanismo de persistencia con Bases de datos

Consultas sensibles a Concerns

Habrá momentos en que deseemos mostrar información diferente para cada concern, y de esta manera facilitar al usuario el uso del sitio a través de navegación más inteligente.

Vamos a utilizar como ejemplo un almacén de instrumentos musicales que permite compras en línea. Imaginemos que identificamos 2 concerns de navegación diferentes, uno para buscar productos por categoría (instrumentos de cuerda) y otro por búsqueda directa mediante el buscador del sitio.

En el primer caso al acceder al detalle del producto tal vez me interese ver otras sugerencias de instrumentos bajo la misma categoría y precio similar, y en el segundo caso seguramente me interese ver sugerencias de accesorios que pueden ser usados con mi instrumento musical.

Además podría existir más de un enriquecimiento en cada caso; suponiendo que planteamos 2 enriquecimientos por concern y que accedemos a ver el detalle de una guitarra "Fender Stratocaster Standard" de U$499, la siguiente matriz resume los resultados deseados.

	a. Por Categoría	**b. Por búsqueda directa**
Listado 1	1.a Instrumentos de igual categoría y similar rango de precio.	1.b Accesorios para el instrumento buscado
	Otras guitarras muy vendidas de precio similar: - Fender Stratocaster Highway One - Ibanez RG 350DX - Dean Vendetta 1	No olvide llevar también: - Cable guitarra LiveWire 1/4" - Pedestal Proline HT1010 - Juego de cuerdas Fender

Listado 2	2.a Instrumentos de igual categoría y que fueron más comprados por otros usuarios.	2.b Otros productos comprados por usuarios que compraron este producto
	Guitarras más vendidas (más de 1000 ventas): - Fender Deluxe Players - Gibson SG Special - Jackson Dinky DR3	Agrega a su compra: - Pod Pocket Express - Correa guitarra Fender - Amplificador Marshal BG 30

Siguiendo este esquema, crearíamos dos vistas (clases que heredan de ViewDefaultState) una para cada concern; las consultas para los listados del primer concern quedarían de la siguiente manera:

```
classProductViewCategoryState extends ProductViewDefaultState{

protected function getFirstListItems(Product $product){

        $category = $product->getCategory()->getId();

        $price = $product->getPrice();

        $rangemin = $price – 150;

        $rangemax = $price + 150;

          $range_between = "between ".$rangemin." AND ".$rangemax;



        $factory = PersistenceAdapter::getInstance();

$products_list = $factory->getObjects("Product","category=".$category." AND price ".$range_between);
```

```
        return $products_list;
    }

protected function getSecondListItems(Product $product){
        $category = $product->getCategory()->getId();
        $factory = PersistenceAdapter::getInstance();
$products_list = $factory->getObjects("Product","category=".$category." AND
solditems> 1000 ");
        return $products_list;
    }
}
```

Estos métodos serán implementados para cada nuevo concern en la clase correspondiente. Alternativamente pueden ser implementados dentro de otro archivo dedicado únicamente a temas de persistencia, es decir en un DAO tal como se usa en otros ORMs. A su vez las consultas serán invocadas desde el método que corresponda al evento de mostrar el detalle del producto.

6.5 Plantillas HTML

Es conveniente y deseable que cada aplicación desarrollada con nuestro framework tenga su propio aspecto gráfico. El sistema además de soportar la característica de crear temas propios para una aplicación, debe permitir que su aspecto sea sensible al concern de navegación activo; es decir que lo que se muestre en pantalla vaya acorde al interés actual del usuario. Sin embargo, no estamos hablando de la información a presentar, ya que este sería el trabajo de cada vista, sino visualmente dónde y cómo se muestra.

Una manera de ver esto es colocar el enriquecimiento visual para cada concern en una matriz de la siguiente manera, tomando como ejemplo el mismo almacén de instrumentos musicales en línea que se usó en el apartado anterior, para dos concerns diferentes:

	a. Por categoría	**b. Por Fecha**
1. Página de inicio	1a. Es deseable navegar por el sitio, seleccionando primero la categoría deseada.	1b. Es deseable conocer los artículos destacados para el día de hoy, para encontrar posibles promociones.
	La página de inicio tiene totalmente a la vista el listado de categorías disponibles, mientras que los artículos destacados están en segundo plano.	La página de inicio tiene en primer plano los artículos destacados, mientras que las categorías disponibles están en segundo plano tal vez en un menú lateral.
2. Detalle del artículo	2a. Al mostrase el artículo mi principal interés es ver otros de la misma categoría.	2b. Al mostrase el artículo mi principal interés es ver otros artículos destacados.
	La página muestra otros artículos importantes de la misma categoría en una zona visible a primera vista, mientras que los artículos destacados se muestran al fondo de la pantalla en segundo plano.	La página muestra otros artículos destacados en una zona visible a primera vista, mientras que los otros artículos importantes de la misma categoría se muestran al fondo de la pantalla en segundo plano.

Internamente se le puede decir al Framework que plantilla se debe usar para un concern específico. La manera más sencilla es tener una carpeta con todos los archivos

HTML que serán usados como plantillas o temas, e indicar en el nombre de cada uno mediante un sufijo, el concern en el que debe ser llamado.

Su implementación es relativamente sencilla, y podría ser incluida en un método de la vista llamado antes de su renderización, donde se tendría que consultar el concern activo y cargar la plantilla correspondiente.

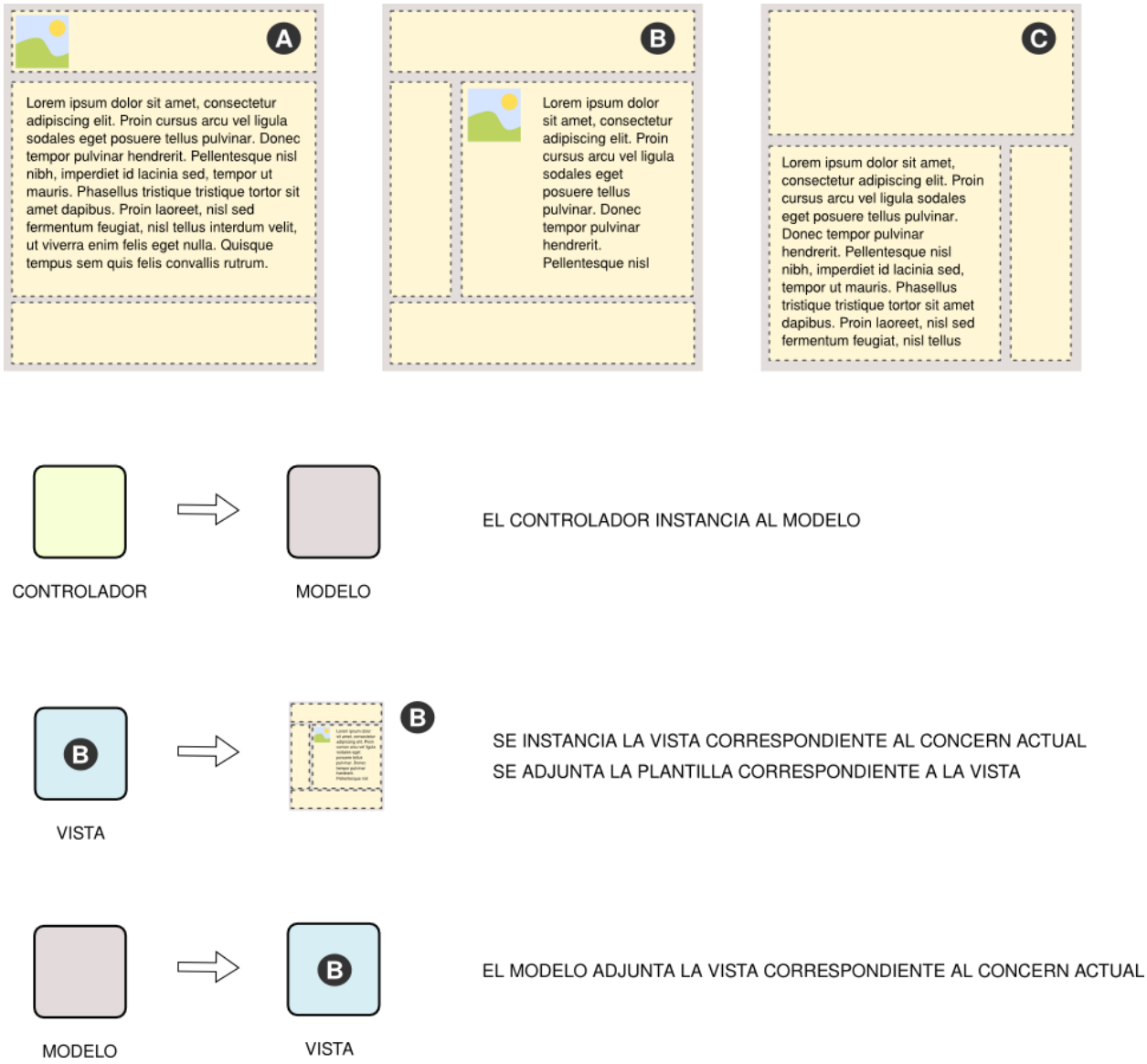


Ilustración 45. Utilización de Plantillas

Podemos realizar este procedimiento desde el constructor de la vista utilizando un poco de reflection para instanciar los objetos necesarios; el código correspondiente a su implementación se muestra en amarillo.

```
classView{
        public function __construct($event){
                $stateName = Reflection::getClassName($this);
                $this->setTemplate($stateName);
        }
}
```

Dentro de cada plantilla existirán marcas que serán reemplazadas por los contenidos generados dinámicamente por el Framework. De esta manera podemos cambiar de una estructura de dos columnas para el concern inicial, a otra de 3 columnas que permita mostrar más información u operaciones para un concern especial.

6.6 Resumen

En este capítulo nos hemos acercado bastante a lo que podría ser la estructura base para la creación de aplicaciones que necesiten ser enriquecidas con CSN, hemos hecho uso de dos patrones ampliamente utilizados como lo son la arquitectura MVC (que a su vez usa Observer), y el patrón State para poder encapsular el comportamiento específico de la aplicación dentro de cada concern.

Además hemos visto a AOP como una alternativa para la notificación de cambios de concern, y así mismo como se pueden usar ACLs y CCLs para consultar el concern activo y sus permisos de acceso. Finalmente se ha agregado al Framework algunos componentes necesarios para complementar el desarrollo rápido de aplicaciones Web de este tipo.

A continuación hagamos una comparación de nuestro Framework con otros existentes, a manera de evaluación, con el fin de identificar aquellos puntos donde se podría trabajar a futuro en cuanto a sus funcionalidades.

	Zend Framework	**CAKE Php**	**Symfony**	**CSN Framework**
Persistencia	ORM con Active Record	ORM con Active Record	ORM (doctrine)	Cualquiera
Navegación Contextual	no	Posible mediante prefijos, pero duplica código innecesario.	no	CSN (statepattern) y componentes que pueden consultar el concern actual.
Cache	XCache	no	APC	No, pero es posible
Ajax	si	si	si	Es posible
MVC	pull	push	push	Pull (observer)
Versión	php5	php4	php5	php5 (OOP pura)

Analicemos solo algunos resultados interesantes:

Persistencia.-

Dos de ellos usan Active Record, lo que quiere decir que las clases de su modelo heredan de una abstracta que contiene métodos para persistir las instancias, esta no es una práctica perfecta, puesto que no separa las instrucciones SQL del modelo en sí. Otro problema de esto, es que necesita que los atributos de las clases, sean los mismos que las columnas de las tablas para que el mapeo sea automático; esto hace que el modelo de clases sea igual al modelo de datos y ya no hablamos de OOP pura al hacer eso.

Sin embargo Symfony usa un ORM llamado doctrine que permite definir las reglas de mapeo en un archivo aparte, evitando "ensuciar" nuestro modelo. Nuestro framework usa una aproximación muy parecida al DAO popularizado en JAVA, lo que permite separar claramente el modelo de los datos. Esto nos permite agregar si queremos un ORM o un data mapper y de esta manera hacer que el modelo no conozca en absoluto sobre persistencia, objetivo deseable en una arquitectura con separación de concerns como la que estamos elaborando.

Navegación Contextual.-

Evidentemente nosotros ganamos claramente aquí. En el resto de frameworks se necesitaría agregar módulos que simulen este comportamiento, o agregar métodos duplicados en los ya controladores existentes. Sin embargo necesitarían modificar el código núcleo de la arquitectura o en su defecto usar técnicas no obstrusivas como AOP o Javascript, pero su adición sería más complicada para lograr que el proceso sea transparente. Nuestra aproximación es mucho más limpia y fácil de implementar, puesto que soportamos CSN de forma nativa considerada en la estructura base de nuestra arquitectura.

MVC.-

Otro aspecto que merece la pena ser explicado sin duda. A pesar de que todos dicen usar un patrón MVC, hemos separado los que lo hacen de manera pura (*pull*) de los otros.

Nos referimos como *pull*, a los que usan el patrón *Observer* para conseguir notificar a la vista los cambios en el modelo. En tanto que los de categoría *push* insertan código extra directamente en el controlador, con el fin de llamar al método de la vista encargado de dibujar la página.
Además necesitan que se le pase el modelo o los datos en variables separadas, con la información necesaria para la renderización. Nosotros hemos preferido la manera tradicional que además nos permite una vez más, una correcta separación de concerns, independizando claramente a los componentes de una triada MVC.

Este análisis nos lleva a pensar que vamos por el camino correcto para lograr nuestro objetivo, el de tener una estructura firme para el desarrollo rápido, organizado y mantenible de aplicaciones Web, que necesiten poseer complejos esquemas de navegación. Esto se verá reflejado a medida que el sistema crezca y necesitemos agregar nuevas funcionalidades.

En el próximo capítulo daremos respuesta a otro de los objetivos de esta tesis, que es la construcción de un gestor de contenidos usando el Framework especificado en este capítulo.

7. Desarrollo de un CMS Prototipo

7.1 Introducción

Como se explicó en los objetivos de este proyecto en el primer capítulo, no solo se pretendía llegar a diseñar el Framework, sino implementarlo y demostrar su utilidad desarrollando una aplicación prototipo con navegación sensible a concerns.

En el capítulo 5 habíamos mencionado que una de las aplicaciones Web que claramente se podían enriquecer con CSN era un sistema de gestión de contenidos. Se plantearon estos enriquecimientos y nos basamos en ellos para darle forma a la arquitectura de nuestro Framework, de modo que pudiéramos utilizarlo como una herramienta base para el desarrollo rápido de aplicaciones similares.

En este capítulo vamos a plantear el diseño de un gestor de contenidos y a comentar sobre los resultados obtenidos luego de su exitosa implementación. Queda claro que se trata únicamente de un prototipo que deja abierta al lector, la posibilidad de agregarle nuevas funcionalidades tanto al CMS como al Framework en sí.

7.2 Diseño

Imaginemos un noticiero o revista en línea que haya sido construido utilizando un CMS, y éste a su vez construido a partir de un Framework con soporte para navegación sensible a concerns. Prácticamente encontramos 2 presentaciones diferentes para los

contenidos en las siguientes formas:En la página de inicio (home) y con la noticia desplegada para leerla completamente.

También podríamos imaginar nuevas presentaciones para resultados de una búsqueda, o para una página de bienvenida después que el usuario haya iniciado sesión. Sin embargo por el momento solo vamos a detallar los enriquecimientos para las dos primeras.

Con esto presente, identificamos en nuestro ejemplo 3 concerns de usuario, o contextos de navegación diferentes.

- Uno por defecto, con el comportamiento habitual para un sitio de este tipo, es decir mostrando las noticias más recientes.
- Otro con las noticias buscadas siempre en base a una categoría especificada por el usuario, aquí el usuario utiliza el menú del sitio para encontrar las noticias de su agrado.
- Finalmente uno para edición, con las funciones y enlaces más utilizados por un administrador. Este esquema de navegación requiere que el usuario haya ingresado previamente sus credenciales e iniciado sesión en el sitio.

Llamaremos a cada concern: **default, category,admin**, respectivamente.

De esta forma, cada presentación es modificada para acoplarse a cada concern. A continuación se muestra una matriz con los enriquecimientos deseados en cada caso. Se debe tomar en cuenta que se podría decir que el concern por defecto NO se encuentra enriquecido en su navegación, y que el resto de concerns heredan este mismo comportamiento, a menos que se indique un enriquecimiento específico.

Concern / Presentación	Home	Dentro de la noticia

Default	1. Un diseño de dos columnas: un listado de las últimas noticias y un listado de las categorías disponibles.	1. Una columna con un listado de las otras noticias del Home.
Category	1. Un diseño alternativo que facilite la navegación por categorías	1. Una columna con un listado de noticias de la misma categoría que la que se visualiza en pantalla. 2. Una opción para cambiar de categoría.
Admin	1. La posibilidad de acceder a secciones del sitio para administrar.	1. Un menú con enlaces para administrar los contenidos del sitio. 2. Enlaces para editar la noticia que se visualiza en pantalla.

Para poder implementar esta idea necesitamos tener en mente los siguientes puntos:

1. El sistema siempre inicia en un concern por defecto. En este caso lo hemos llamado Default y no contiene enriquecimientos específicos, sino más bien el esquema general de navegación. Debe existir una manera de poder regresar luego a él.
2. Para poderle indicar a la aplicación el cambio de concern, es necesario establecer ciertos eventos que disparan el cambio, es decir cuando el usuario hace clic en algún enlace específico (punto de cambio) esto desencadena el reemplazo del concern activo.
3. Puede existir más de un punto de cambio que nos lleve al mismo concern, es decir varios enlaces a lo largo del sitio que tengan como destino un mismo esquema de navegación.
4. Puede haber concerns que necesiten que el usuario inicie sesión, puesto que manejamos un sistema de permisos que permite especificar los objetos

(controladores, acciones) a los que un rol puede acceder en determinado concern.

Además de los puntos anteriores, debemos tener en cuenta que el usuario seguirá navegando por el sitio en el nuevo concern, hasta que pase nuevamente por otro punto de cambio, que lo lleve hacia otro.

Para no provocar confusión en el usuario debe ser fácil de cambiar de un concern a otro y de indicarle su esquema actual, entonces debemos considerar los siguientes requisitos:

- El sistema debe ser capaz de recordar el último concern de interés del usuario, para que se retome el mismo esquema cuando éste regrese al sitio nuevamente.
- Al entrar a la página de inicio, deben existir tantos puntos de cambio como cantidad de concerns de navegación.
- También es necesario un ícono o señalización en pantalla que nos indique el concern en que nos encontramos actualmente.
- Es necesario indicar las zonas que se verán afectadas por el cambio de concern. En la matriz anterior estas zonas quedaron establecidas por los números 1 y 2 cuando listamos los enriquecimientos posibles. Por ejemplo, cuando empezamos en el concern Default y vamos adentro de una noticia, la zona 1 mostrará una columna con un listado de las otras noticias del Home.
- Si cambiamos de concern (category) y regresamos al mismo lugar ahora se debería mostrar en la misma zona 1, un listado de noticias que pertenecen a la misma categoría que la noticia que se visualiza en pantalla.
- Y si finalmente cambiamos nuevamente de concern y regresamos a la noticia desplegada, se debería mostrar ahora un menú con enlaces para administrar los contenidos del sitio.

En la **¡Error! No se encuentra el origen de la referencia.**, se nos presenta la página de inicio en el concern Default, es decir tal como se ve cuando entramos al sitio

por primera vez. Aquí se puede observar los puntos de cambio hacia cada concern, y además una señalización para el concern actual, que consta de un ícono semáforo con los siguientes valores:

Concern/Color	
Default	Verde
Category	Amarillo
Admin	Rojo

Ilustración 46. Página de inicio - concern por defecto

Si accedemos al concernCategory a través de uno de sus puntos de cambio y luego queremos volver a la página de inicio sin regresar nuevamente al concern por defecto, hacemos clic sobre el logo del sitio y tendremos una pantalla como se muestra en la **¡Error! No se encuentra el origen de la referencia.**. Se debe tener en mente que si hacemos clic sobre el ícono de home que se encuentra en la parte superior, este no solo nos llevará a la página de inicio, sino que nos cambiará al concern por defecto ya que se encuentra marcado como punto de cambio.

Como se puede apreciar en la **¡Error! No se encuentra el origen de la referencia.**, la página se ha reconfigurado para ser navegada por categorías, y también como se esperaba se encuentran visibles ciertos puntos de cambio para navegar a través del resto de concerns. Además podemos observar que nuestro semáforo ha cambiado a color amarillo para indicar el cambio al concernCategory.

Ilustración 47. Página de inicio - concernCategory

Si accedemos al home a través del concernAdmin, también notaremos como la página se acopla a este, permitiendo una navegación más acorde a los intereses de un administrador de contenidos.

Ahora veamos un ejemplo de cómo se mostraría la noticia desplegada completamente, para cada uno de los concerns.

Ilustración 48. Noticia desplegada para el concern Default

Para el concern por defecto, podemos observar que la disposición de los elementos en la página ha formado dos columnas, una principal con detalles únicamente de la noticia desplegada que es del interés por el usuario, y otra columna que contiene un listado de

otras noticias actuales e importantes ya que aparecen en portada. Este esquema de navegación sugiere que el usuario debería revisar las otras noticias destacadas para el día de hoy. Debemos notar que el semáforo es de color verde indicando el concern actual.

Ilustración 49. Noticia desplegada para el concernCategory

Para el concernCategory, la diferencia es clara. Más allá del orden de las columnas, ahora la principal a la derecha y el listado de relacionadas a la izquierda; se puede observar que las sugerencias para el usuario incluyen únicamente noticias que se encuentran bajo la misma categoría. Esto se debe a que nuestro visitante ha decidido explorar los contenidos de acuerdo a la categoría de su interés, por lo que lo común es que luego de revisar la actual quiera seguir viendo otras dentro de la misma temática.

De igual manera que el concern anterior, mostrará primero las más recientes pues nos interesa mostrarle noticias actuales a final de cuentas. Finalmente nótese que el semáforo es de color naranja para indicar el concern actual.

Ilustración 50. Noticia desplegada para el concernAdmin

En esta ocasión la transformación del esquema de navegación es profunda. Tenemos nuevos enlaces tanto en la columna principal como en la lateral, los primeros nos llevan a realizar modificaciones sobre la noticia desplegada y los de la columna lateral nos permiten visitar otras páginas con tareas de administración frecuentes. La disposición

de los elementos a través de la plantilla correspondiente a este concern, permite tener los enlaces en una región bajo la noticia desplegada lo que en otros concerns no era posible colocar.

Aunque no lo hemos mencionado antes, pero resulta evidente que para poder acceder a un nodo desde un concern de administración es necesario tener en la mitad una página que pida credenciales al usuario; solo después de validarlas, el usuario podrá acceder al nuevo concern.

La seguridad se valida mediante ACLs para decidir aquellos roles que tienen permiso sobre ciertos objetos. En realidad con la arquitectura definida los permisos son mucho más fáciles de manejar, ya que CSN hace que no se requiera una página diferente para modificar un contenido. Por lo que más que especificar una a una las páginas permitidas, lo que se valida es si el usuario puede cambiar inmediatamente a un nuevo concern al pasar por una URL de cambio.

7.3 Resultados

Entonces en síntesis, los enriquecimientos en nuestro ejemplo fueron los siguientes:

- los contenidos específicos para cada concern que se muestran en una misma zona de la página como si fueran sugerencias. Esto se aprecia en el listado de noticias que acompañaba a un noticia desplegada, el cual era diferente para ajustarse a los intereses del usuario
- la disposición de los elementos en la páginaque era variada, gracias a que cada concern posee una plantilla HTML propia
- las funciones u operaciones disponibles sobre el contenido como en el caso del concern de administración, que permiten realizar acciones rápidas sin tener que explorar un menú o página por separado.

Luego de la implementación no solo se evidencia que fue posible obtener la aplicación que deseábamos, sino se evidenció la buena organización de código en el Framework,

que no limitó de ninguna manera su realización. La facilidad para incluir un nuevo concern es grande; solo basta con crear una nueva clase para la vista de una triada MVC, heredar el comportamiento del concern Default y realizar los cambios visuales necesarios.

Las decisiones sobre aquellos enlaces que provocan un cambio de concern se manejó a nivel de base de datos, lo que permitió tener una interfaz para indicar una URL del cambio y el concern de destino.

El aspecto de seguridad fue fácil de manejar, gracias a la capacidad del CMS para otorgar o negar permisos a un rol (en nuestro caso anónimo y administrador) hacia una acción específica.

La variedad de aplicaciones para la navegación sensible a concerns es realmente grande. En nuestro ejemplo para un sitio web de noticiero sobre un gestor de contenidos, es aún larga la lista de enriquecimientos que podemos agregar; ahora para cualquier otro tipo de sitio web es fácil imaginar una variedad de usos y de mejoras en la experiencia del usuario final. En el siguiente capítulo se mencionan algunos trabajos futuros en esta misma dirección.

8. Trabajos Futuros

Luego de la implementación del gestor de contenidos, nos damos cuenta que queda mucho por hacer tanto para otras aplicaciones de CSN como para fortalecer el Framework desarrollado. De las ideas más importantes sobre lo que queda pendiente de este trabajo tenemos las siguientes propuestas:

8.1 Agregar inteligencia al mecanismo de reconocimiento de cambios de concern a lo largo de la aplicación

Nosotros hemos establecido una manera de notificar a la aplicación sobre los cambio de concern a partir del mapeo de las URLs que definen cada página del sitio Web. Así, si un usuario visitaba una página incluida en nuestro listado, el sistema sabía a qué concern debía cambiar antes de mostrar la página, y no cambiaría al próximo hasta encontrarse con otra URL conocida.

El problema de este método es que el programador tiene que realizar el mapeo para cada una de las páginas del sitio Web, y si éste es demasiado grande la tarea es extensa y propensa a errores.

Una mejor alternativa sería aplicar inteligencia artificial para dejar que la aplicación infiera cual debería ser el concern activo para una página visitada. Se podría utilizar un conjunto de reglas a partir de las cuales realizar un mapeo automático, o aún mejor podríamos usar ontologías para relacionar todo concern dentro de la aplicación y usar alguna herramienta que se encargue de interpretarlas. La ventaja sería que el ingeniero solo tendría que diseñar el árbol de conceptos una vez y a partir de la actividad del usuario concluir que concern se debería usar en ese momento.

La introducción de este motor de inferencia semántica no compromete la arquitectura de nuestro Framework, dado que solo se debería intercambiar aquel componente que se encargaba del mapeo de URLs a concerns.

8.2 Plantear los cambios necesarios para permitir la combinación de varios concerns activos

Cuando hablamos de concerns de navegación, hablamos de diferentes esquemas para los enlaces de la aplicación en base a un contexto de uso. Podemos deducir que nuestro trabajo cae en el mismo ámbito de las aplicaciones sensibles al contexto, concebidas para brindar una personalización a la aplicación en base a la adquisición de información a partir del contexto actual. Tomando en cuenta la misma información que utilizan estas aplicaciones en base a estas 4 categorías: identidad, actividad, tiempo y ubicación del usuario; vemos que se pueden combinar estas condiciones para inferir cuál es el verdadero interés del usuario en un momento dado.

De la misma manera podríamos querer inferir el concern activo, a partir de más de un parámetro. Por ejemplo nos interesaría mostrar un esquema de navegación específico para cierto usuario que visita nuestra página a cierta hora del día, de cual conocemos también su actividad anterior en el sitio y el lugar donde vive.

Sin embargo, en lugar de realizar infinitos esquemas de navegación para cada una de las combinaciones que pudiéramos hallar, sería mejor tener un mecanismo para combinar dos o más esquemas existentes. Es decir, si tenemos por ejemplo ya un esquema para los usuarios que viven en Latinoamérica, otro para los usuarios que son potenciales compradores (tal vez porque tienen ítems en su carro de compras) y otro para los que navegan el sitio realizando búsquedas directas; se podrían compartir los enlaces de cada uno y mostrarlos en pantalla en regiones diferentes, o filtrarlos de forma inteligente para mostrar solo los que no sean redundantes.

Esta propuesta se puede combinar con la anterior del motor semántico para lograr un mecanismo de enriquecimiento único, el cual sería parte importante de nuestro framework para crear aplicaciones Web realmente impresionantes.

9. Conclusiones generales

Al final de este trabajo podemos confirmar que las aplicaciones Web 2.0 actuales pueden enriquecerse fácilmente para permitir una arquitectura escalable, y al mismo tiempo mejorar la experiencia del usuario final.

Se ha insistido a lo largo del documento en que una de las ayudas más notables para el usuario es perfeccionar la estructura de navegación y por tanto los enlaces que se le muestran en un momento dado. La experiencia del usuario mejora notablemente al proveerle de mejor información mientras navega por una página, lo cual implica ayudarle a hallar lo que necesita agregando la mayor cantidad de enlaces útiles hacia páginas con información relacionada con sus intereses.

De esta manera estaremos sugiriendo la ruta que debe seguir para encontrar lo que busca de una manera más rápida, en lugar de tener una cantidad excesiva de información confusa en la página de inicio como sucede en sitios convencionales. Esto quiere decir que no se trata de poner a su disposición todos los menús y enlaces posibles hacia todas las partes del sitio Web, sino de inteligentemente mostrarle los caminos adecuados para llegar a la página que desea ver basándonos en su actividad previa.

La definición de una estructura base que permita el enriquecimiento de aplicaciones por medio de concerns de navegación, es posible por medio de la combinación de patrones de diseño, de patrones de arquitectura y de técnicas actuales en el desarrollo de aplicaciones Web modernas. Prueba de ello es el diseño que obtuvimos para el Framework que construimos, el cual armonizaba el MVC con patrones como el State, el Adapter, el Singleton, entre otros; además de mapeadores objeto – relacional, programación orientada a aspectos, archivos XML y muchos más. Lo más importante es que nuestro acercamiento es totalmente orientado a objetos, lo que apoya la reutilización de código, la extensión de la arquitectura mediante herencia y más.

Aún cuando al escuchar el gran número de herramientas que se utilizaron, el diseño y su implementación fueron relativamente sencillas y podríamos estar seguros de que no es necesario comenzar desde cero como lo hicimos, sino que podríamos agregar un módulo o componente que implemente CSN a uno de los frameworks comerciales existentes en cualquier lenguaje de programación.

Además tal como mencionamos en el capítulo anterior cuando hablamos de los trabajos futuros, es posible mejorar nuestra propuesta agregando cierta "inteligencia" a la parte del framework que debe decidir qué concern se debe usar en un momento dado de la aplicación.

Finalmente sabemos que los gestores de contenido no son las únicas aplicaciones que pueden verse enriquecidas con CSN, y por lo tanto construidas de forma ordenada con nuestro framework. Otros de los trabajos futuros que podemos agregar a nuestra lista es el análisis de cómo enriquecer el resto de aplicaciones Web 2.0.

10. Referencias

1. Cameron Chapman. "TheSmashing Magazine.COM": 15 Errores comunes en diseño E-Commerce. Accedida en Marzo de 2010
2. Tyler Tate. TheSmashing Magazine.COM": Minimizando la Complejidad en interfaces de usuario. Accedida en Marzo de 2010.
3. Nigel Bevan. Problemas de Usabilidad en Diseño de Sitios Web (1998), Guías y Estándares de usabilidad Web. NationalPhysicalLaboratory, UsabilityServices (2005).
4. Mario Sánchez, AlteredImpressionswebsite: Consejos de Usabilidad - Navegación en Sitios Web: La analogía del Centro Comercial(2003).
5. www.albanova.com El Portal de los Negocios en Internet. La Navegabilidad de la Web. Accedida en Abril de 2010.
6. MaishNichani, PebbleRoad. Improving the User Experience with In-Page Navigation. 2007

7. Jonathan Lazar. Web usability: a user-centered design approach. Pearson Addison Wesley, 2006
8. James Kalbach. Designing Web Navigation. O'Reilly 2007.
9. J.English, M.Hearst, R.Sinha, K.Swearington, P.Yee. Examining the Usability of Web Site Search. University of California, Berkeley. 2001
10. Catalyst Group Design. "Net Rage" A Study of Blogs and Usability. 2005. Disponible en www.catalystgroupdesign.com
11. Jeffrey Zeldman. Designing with Web standards. New Riders Publishing 2003
12. SanJay J. Koyanl, R. Balley. Research-Based Web Design & Usability Guidelines. 2002
13. VLADIMIR O. SAFONOV .Using Aspect-Oriented Programming for Trustworthy Software Development. Wiley2008
14. Artículo de Wikipedia acerca de: Separation of concerns (SOC). Disponible en http://en.wikipedia.org/wiki/Separation_of_concerns
15. Derek Greer, Ctrl-Shift-B website: Artículos sobre arquitectura de software y desarrollo. El arte de la separación de Concerns. Accedida en Diciembre de 2010.
16. Nanard, J., Rossi, G., Nanard, M., Gordillo, S., Perez, L. Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns.
17. AspectOriented Programming (AOP) Library for PHP. Una librería para implementar AOP desde PHP. http://www.aophp.net/index.php. Accedida en Mayo de 2010.
18. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications with OOHDM.
19. Margarita Figueroa, Earth Council. Navigation Schemes: Issues and Concerns. Mayo de 2001
20. Amy Shuen. Web 2.0: A Strategy Guide. O'Reilly (2008)
21. Nanard, J., Rossi, G., Nanard, M. Engineering Web Applications using Roles. Universidad de la Plata

22. Nan Niu, Steve Easterbrook, Yijun Yu. Taxonomy of Asymmetric Requirements Aspects. Department of Computer Science, University of Toronto
23. Awais Rashid and Yijun Yu. Aspect-Oriented Requirements Engineering: An Introduction (2008).
24. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US, 1994.
25. Stefano Ceri, Piero Fraternali, Aldo Bongio.Web Modeling Language (WebML): a modeling language for designing Web sites. Dipartimento di Elettronica e Informazione, Politecnico di Milano.
26. H. Casalánguida, J.E. Durán. Aspect Oriented Navigation Modeling for Web Applications Based on UML. IEEE LATIN AMERICA TRANSACTIONS (2009)
27. Artículo de Wikipedia acerca de: Context-sensitivehelp. Disponible en : http://en.wikipedia.org/wiki/Context-sensitive_help
28. Steve Burbeck. Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC). 1992
29. Peter Lavin. OBJECT-ORIENTED PHP Concepts, Techniques and Code. 2006
30. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern Oriented software architecture, a system of patterns. 1996
31. E. Gamma, R. Helm, R. Johnson and J.Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. 1994
32. Daniel M. Brandon. Software Engineering for Modern Web Applications: Methodologies and Technologies.
33. Judith Bishop. C# 3.0 Design Patterns. O´Reilly 2007
34. James Governor. Web 2.0 Architectures. O' Reilly 2009
35. Martin Fowler. Patterns of enterprise application architecture. Addison-Wesley Professional, 2003
36. Gary Mailer, Michael K. Glass, Jason Gerner, Mike Glass, Jeremy Stolz, Yann Le Scouarnec, Elizabeth Naramore. ECM/CMS: Content Managements. 2008

37. Forrest Lyman. Pro Zend Framework Techniques: Build a Full CMS Project. Apress 2009.
38. Matt Zandstra. Php objects, patterns, and practice, Apress 2008