FACULTAD DE INFORMATICA
UNLP

Titulo: “Aplicaciones ricas en Internet (RIA). Un enfoque de refactorizacion”

Autor: Eliseo Palacios
Director: Dr. Gustavo Rossi
Codirector:

Resumen: Las páginas Web actuales carecen de cierta performance, son poco interactivas, lo que hace que en términos generales las aplicaciones sean difíciles de usar, hay muchos aspectos que podrían ser mejorados para mejorar la interactividad, la navegabilidad, la performance entre otros.
Estudiaremos el concepto de aplicaciones RIA (Rich Internet Applications), Características principales, como mejora este el grado de interacción de un usuario con la aplicación, que tecnologías utiliza una aplicación con estas características, con que herramientas desarrollar, como llevar una aplicación legacy a una aplicación RIA, que rol cumple AJAX en una aplicación RIA. Estudiar el concepto de “refactoring” para el enriquecimiento del diseño de datos y con esto darle una mayor escalabilidad y no hacer tan dura la tarea de modificar una parte del código, de una aplicación convencional.
Analizaremos de qué manera podríamos utilizar el concepto “Refactoring” en una aplicación RIA tanto a nivel navegacional como a nivel de presentación.
Identificar y analizar una serie de pequeños refactoring para aplicar en los niveles mencionados para enriquecer una aplicación Web.

Líneas de Investigación: Ingenieria Web

Trabajos Realizados: Estudiamos una aplicación legacy y aplicando nuestra serie de refactorings desarrollados llegamos a una aplicacion con características mas atractivas, mas eficientes, mas confiables, llegamos a una aplicación con características mas “ricas”

Conclusiones: Estudiamos las características de las aplicaciones Web tradicionales, con ello vimos que la mayoría de estas eran falencias que poseen las mismas.
Un tiempo mas tarde estudiamos las características de las Aplicaciones Ricas en Internet, su arquitectura, como difiere un request en una aplicación “rica” y en una aplicación tradicional, también vimos como evitar las sucesivas recargas de las páginas, las diferencias de performance entre unas y otras.
En el tramo final analizamos el concepto de refactoring, y como podemos aplicarlo a las aplicaciones Web, propusimos una serie de refactoring para poder mejorar ciertos aspectos de una aplicación Web tradicional y acercar su performance a las aplicaciones ricas.
Teniendo en cuenta estos podemos solucionar varios inconvenientes que poseen los usuarios al utilizar una aplicación.
Llegamos a la conclusión que podemos llevar nuestra aplicación Web hacia una aplicación “rica”, haciendo pequeños cambios, ósea, haciendo sucesivos refactoring sobre las funcionalidades, sobre la capa de presentación y/o sobre la la capa de navegación de la aplicación.

Trabajos Futuros: Se esta trabajando en tener una herramienta de diseño web que pueda contemplar un cambio en el modelo navegacional, por ejemplo, o en el modelo de aplicación y mapear esos cambios al modelo navegacional y/o a nivel de presentación si el refactoring lo requiere

Fecha de presentación: Diciembre de 2008

1. Introducción

Con el correr del tiempo se ha hecho tedioso el entrar a un sitio Web en busca de información, enviar un mail, querer comprar algo, customizar una aplicación, etc.

¿Porque se ha hecho tedioso?

Si bien las aplicaciones Web cobraron mucha importancia en los años 90, en esta última década las aplicaciones de e-commerse fueron las que mas se destacaron.

Es amplio el espectro del tipo de aplicaciones que hoy por hoy existen en la red, estas últimas y los grandes gestores de mails son las de mayor concurrencia. Entre ellos podemos nombrar Google, Hotmail, Yahoo, eBay, Amazon, etc.

Como dijimos antes, la mayor concurrencia de usuarios se dan sobre este tipo de aplicaciones, pero el desazón de los usuarios también es mucha ya que la mayoría de ellas son difíciles de manejar, poco entendibles, con poca interactividad, lo que concluye en una mala experiencia del usuario para con la aplicación.

Hoy en día hay ciertas compañías que se han ido dando cuenta de esto, y basados en eso, y en lo que el usuario de Internet (como algunos la llaman) necesita han logrado llegar a tener otro punto de vista y así empezar a encaminar su compañía.
Las grandes compañías que nombramos en el párrafo anterior, ya tienen online sus aplicaciones RIA, Hotmail, ha migrado toda su aplicación y hoy cuando entramos a chequear nuestros mails podemos hacerlo tanto desde la aplicación RIA como desde la aplicación tradicional.
Yahoo opto por aplicar la misma técnica, mantener online ambas aplicaciones, por un determinado tiempo, hasta que el usuario logre adaptarse a los cambios. Un cambio muy radical por lo general produce una alteración en el socio, estudios han comprobado que si el usuario se encuentra perdido, literalmente, dentro de la aplicación puede llevar al alejamiento del mismo.

Migrar una aplicación por completo requiere recursos humanos y periodos de desarrollo directamente proporcional al tamaño de la aplicación.

Creemos que podríamos solucionar el problema de una migración grande realizando el mismo, pero de a pequeños pasos.
Estos pequeños pasos no deberían alterar el resultado final de la función que estemos retocando, solo mejorar la forma y darle seguridad al usuario sobre la acción que se encuentra realizando.

La idea de esto es llevar la Web hacia una nueva generación con la misma funcionalidad, pero de mejor forma, más eficiente, más confiable, más confortable, etc. Es decir a grandes rasgos mejorar la interacción del usuario

con la aplicación, como por ejemplo dándole respuestas inmediatas ante una acción ejecutada, ayudándolo de forma inmediata (online) al querer realizar una acción, etc.

Otra desventaja importante a tener en cuenta son las sucesivas recargas de pagina cada vez que ingresamos a un sitio la pagina tenia grandes periodos de recarga (mas de lo que la necesitaba), en la actualidad esto esta cambiando gracias la integración de tecnologías, que si bien existían, pocos hacían uso de ellas de manera conjunta para poder lograr grandes resultados, esto tiempo después se conoció con el nombre AJAX.

Hay compañías que han ido más allá de migrar sus propias aplicaciones, y han apostado un poco mas, Google fue mas allá del desarrollo de su propio gestor email, desarrollo un Framework de desarrollo de aplicaciones ricas en Internet.

Esta tipo tecnología ayuda tanto al programador como al usuario de aplicaciones Web, ya que no ve tanto tiempo la pagina en blanco (recarga), y el programador ahorra grandes cantidades de tiempo para desarrollar aplicaciones con características realmente atractivas.

Pensamos que no son necesarios grandes cambios para llevar nuestra aplicación a RIA, con pequeños cambios bien focalizados podemos llegar al mismo resultado, obtener aplicaciones “Ricas”, más amigables, confiables, informativas, en fin usables.

El único fin de todo esto es darle al usuario otra visión de la aplicación y mejores resultados con cada acción que realiza, en fin las compañías han entrado en razón y quieren recuperar algo lo mas importante, los usuarios.

Teniendo en cuenta cuales son las principales falencias de las aplicaciones Web de hoy en día, lo que el usuario desea obtener cuando ingresa a una aplicación Web, cuales son las cosas que le resultan molestas al momento de realizar una acción, cuales serian las posibles soluciones, intentaremos encontrar proponer una serie de cambios que podemos hacer sobre las aplicaciones para que sean mas productivas.

Para ello estudiaremos las principales características de las aplicaciones ricas en Internet, beneficios de desarrollar una aplicación rica, arquitectura, comparaciones entre el modelo de una aplicación Web tradicional y una “rica”, herramientas de desarrollo, AJAX, el rol que cumple dentro de las mismas

Por ultimo intentaremos encontrar un punto de vista de cómo hacer para llegar a brindarle al usuario una aplicación “rica” partiendo de una aplicación Web tradicional interponiendo los refactorings.

El concepto de refactoring hace hincapié en el diseño de las aplicaciones, en el modelo de objetos, en la capa de aplicación de una aplicación Web, nosotros estudiaremos pequeños cambios que podríamos hacer sobre las aplicaciones Web tradicionales que faciliten a los usuarios las tareas, cambios que hacen que las aplicaciones Web sean más interactivas y con una mejor performance.

Intentaremos aplicar el concepto de refactoring en las aplicaciones Web, esto es, con pequeños cambios podríamos mejorar la forma de las funcionalidades sin desviarnos del resultado final de las mismas.

De esta manera nos focalizamos en pequeños objetivos, trabajamos sobre ellos, y ponemos en producción los mismos, luego iniciamos el mismo proceso con otra parte de la aplicación, y así sucesivamente.

De esta manera podemos encarar nuestro proceso de “enriquecimiento” de una aplicación desde un punto de vista mas practico sin necesidad de encarar un nuevo desarrollo.

2. La Web de hoy en día (WEB 1.0)

Normalmente en las aplicaciones Web, hay una recarga continua de páginas cada vez que el usuario pulsa sobre un link o un botón. De esta forma se produce un tráfico de datos muy alto entre el cliente y el servidor, haciendo que se recargue la página por completo así tenga que refrescar el valor de un simple input.

Otra de las desventajas de las tradicionales aplicaciones Web es la poca capacidad multimedia que posee. Para ver un vídeo tenemos que usar un programa externo para su reproducción.

2.1. Características

Con el avance de la tecnología y las redes de comunicación las aplicaciones Web emergieron rápidamente hacia una posición dominante en el mercado.

Tanto las capacidades de las tecnologías cliente, como las tecnologías de servidor, para procesar una pagina Web, han sido enriquecidas, optimizadas, estandarizadas, en definitiva hoy en día son mas potentes, dinámicas e integrables con aplicaciones de back-end.
Así mismo la demanda del usuario llevo a un crecimiento continuo que fue sobrepasando las capacidades que las tecnologías nos podían dar al momento del desarrollo.

Estudios sobre usuarios de aplicaciones de e-commerse demostraron que los más altos porcentajes de disconformidad se dan sobre:

- Dificultad de navegación
- Lo difícil que se hace encontrar un producto sobre un sitio de estas características
- las malas experiencias en la compra de productos online

Teniendo en cuenta todo esto, las aplicaciones deberían ser más fáciles de navegar, proveer un método mucho más sencillo y fácil, por ejemplo, para realizar una compra en aplicaciones de e-commerse.
Esto es solo un ejemplo a modo de demostración de que es lo que hace que el usuario no utilice, o mejor dicho tenga desconfianza al momento de usar una aplicación Web, esto mismo ocurre en la mayoría de las aplicaciones Web, el usuario no tiene la misma confianza que si realizara la tarea sobre una aplicación de escritorio.
Hoy en día el principal reto de las aplicaciones es adaptarse a la forma en que se deben mostrar las cosas de acuerdo a la complejidad de la aplicación

Representar una funcionalidad que requiere tres o cuatro pasos podrían ser representados en una sola página, sin embargo debido a las limitaciones de interactividad de HTML, lo haría probablemente en tres pasos que confundirían, lo harían poco manejable o demasiado largo para un usuario que necesita que sea los mas fácil posible de realizar.

La solución mas sencilla es partir la tarea en 3 o 4 pasos y típicamente mapearlas a 4 paginas nuevas, haciendo larga la forma en que el usuario debe completar la tarea, yendo de pagina en pagina hacia adelante y hacia atrás creando en el usuario interrupciones que concluyen en un resultado lento, torpe, y confuso y sobre todo una experiencia frustrada.

Lo peor de todo esto es que los usuarios de aplicaciones Web han sido convencidos de que:

- todas las tareas ahora deben ser multi-pasos
- todas las tareas deben ser en un flujo de varias paginas
- es necesario navegar hacia otra pagina para ver los resultados de la búsqueda que el usuarios selecciono, en vez de mostrarlos sobre la misma pagina sobre la que selecciono los criterios
- no indicarnos que un link lleva a una pagina con cero resultados.

Lamentablemente los usuarios han sido convencidos de que todos estos ejemplos son los mejor para llevar a cabo una tarea.

Varias paginas para una tarea que requiere mas de un paso, no esta mal, pero estamos necesitando una nueva representación para este tipo de procesos, que reduzca el numero de pasos, transiciones de pagina, mejorando el grado de interacción del usuario, en muchos casos podríamos representar estos mismos 4 pasos en una sola pantalla interactiva sin necesidad de muchas transiciones y mucho mas intuitivo.

Hoy por hoy muchos programadores siguen utilizando técnicas antiguas al momento de programar, y muchos otros no tienen la formación suficiente para desarrollar.

Desarrollar un sitio Web es una gran tentación para cualquiera, ya que con cualquier editor de texto o aplicaciones del estilo de Dreamweaver cualquiera puede poner online un sitio Web sin tener la menor noción de que es “request”.

2.2. Limitaciones

2.2.1. Problemas de customización

Con el paso del tiempo las empresas han echo grandes esfuerzos por permitirle al usuario customizar la forma en que ven las secciones en los sitios, la forma en que se le ofrecen los productos, la forma en que se le comunican ofertas.
Sin embargo proveerle al usuario esta funcionalidad es muy complejo, ya que hay que mostrarle al usuario las combinaciones que hacen a su perfil, seleccionándolas de un centenar de posibilidades. Además todas las posibles combinaciones deberán ser validas.

La solución a esta complejidad debería satisfacer los siguientes puntos:

- indicar el estado de los productos seleccionados
- indicarle al usuario que combinaciones son validas y cuales invalidas
- indicarle al usuario que elementos seleccionados causan problemas y cual seria la solución
- indicarle al usuario los costos individuales de cada producto y los totales

2.2.2. Diseño pobre de datos

Por lo general en las aplicaciones Web en lo último que se hace hincapié es en el modelo de datos y la manera en que estos se relacionan.
La falta de una modelo de datos y de un modelo navegacional hace que las aplicaciones sean difícil de navegar y ayuda a las malas experiencias de los usuarios.
La falta de uso de patrones lleva a un modelo pobre de datos.
Un buen diseño lleva a un buen modelo de navegación y un buen modelo de navegación es esencial para una capa de presentación amigable.
Por lo tanto seria ideal aplicar patrones de diseño al momento de diseñar las aplicaciones Web, darle mas importancia a esta capa para obtener mejores resultados.
La falta de normalización hace a un modelo navegacional sin sentido, que nos llevaría a navegar por lugares por los cuales no deseamos transitar en nuestra experiencia con la aplicación.

2.2.3. Complejidad de las búsquedas

En las aplicaciones Web actuales tal como dijimos anteriormente, la tarea de encontrar el producto adecuado es muy complicada, supongamos que queremos buscar una cámara de fotos debemos ingresar criterios de búsquedas (pixeles, flash, precio, etc.), que en la mayoría de las aplicaciones son textuales. Y al usuario se le presentan en una página en forma de listado para que el usuario encuentre la que mejor se adapte a los criterios de búsqueda.

Este tipo de problemas se podría solucionar si al usuario se le presenta una listado de cámaras en forma grafica y que el usuario pueda seleccionar que tipo de cámara de fotos desea, a su ves acompañar a esta funcionalidad de búsqueda con checkbox con las características que pueden tener las cámaras de fotos, o bien podría ser un listbox con multi-selección para poder seleccionar los criterios.

De esta manera el usuario al usuario solo se le presentara un listado de imágenes de cámaras de fotos que cumplan con los criterios que el selecciono.

Permitirle al usuario interactuar con la aplicación y ver los productos que el desea de manera casi instantánea, esto concluye con una buena experiencia del usuario con la aplicación Web.

2.2.4. Ausencia de Feedback

Estamos necesitando que las aplicaciones sean mucho mas interactivas e inteligentes para que puedan darle al usuario una respuesta ante un cambio en el estado de sus datos, un ejemplo puede ser cuando enviamos un email o bien cuando marcamos un email como leído, sin necesidad de que paginas completas sean refrescados de estado y largas comunicaciones con el servidor.

Teniendo en cuanta que hoy por hoy el grado de cohesión entre el servidor y el cliente ha disminuido podemos enfocarnos en crear experiencias de usuario donde se trate el problema de la falta de feedback en las aplicaciones Web.
Este es un punto a tener en cuanta al momento de desarrollar una aplicación Web.

2.3. Las Limitaciones de las paginas HTML

Una de las claves por las que las aplicaciones Web actuales no pueden solucionar los problemas antes mencionados es por las limitaciones de las paginas HTML.

Internet se basa en clientes tontos acoplados ligeramente con servidor que son los que proveen la inteligencia en esta estructura y reciben los request de los clientes, por esto mismo una gran cantidad de retos de las aplicaciones Web son atribuibles a la arquitectura actual de un modelo done los clientes no tienen inteligencia.

En este modelo la página es la interfase por defecto para el usuario
Esto define la interfaz de usuario y el grado interactividad, esto es también la unidad mas pequeña de transferencia y refresco. Esto es autocontenido Y por lo tanto requiere que un mecanismo externo mantenga la información del contexto a través de páginas

Finalmente, esto es una división de procesos natural y artificial, este es el mecanismo usado para realizar los request de nuevos contenidos o enviar una respuesta, o mejor dicho la forma de segmentar el proceso en pasos.

El uso de este modelo limitado ha llevado a desarrollar una variedad de algoritmos sofisticados de cachee. Mecanismos y prácticas de código cuya principal finalidad era agilizar las recargas de páginas y reducir el uso del ancho de banda.
Si bien agilizar la muestra de contenidos se logro, aun resta solucionar la falta de interacción de los cambios de contenidos con el usuario.

Necesitamos encontrar que el cliente de ser un objeto completamente dependiente del servidor y cobre autonomía y pueda manejar los requerimientos de cliente a su gusto, por ejemplo de manera asincrónica. Por lo general debe esperar una respuesta del servidor para poder enviarle al usuario una respuesta, si bien eso no esta mal, el servidor muchas veces no puede darle una respuesta inmediata, por eso seria indispensable que el cliente deje de ser solo un pasamanos de datos.

Por todas estas características antes nombradas esta arquitectura se ha vuelto difícil de manejar y estamos necesitando nuevos horizontes para poder darle al usuario nuevas características mucho más confiables y amigables, que sirvan para mejorar la productividad, dejen satisfecho al usuario.

3. La nueva generación de aplicaciones Web - Web 2.0

El concepto original de la Web (en este contexto, llamada Web 1.0) eran páginas estáticas HTML que no eran actualizadas frecuentemente.
El éxito de las paginas Web dependía de la creación dinámica de HTML creados al vuelo desde una actualizada base de datos, Hay dos cosas que eran muy importantes, que el sitio sea visitado y el lock & feel.
Los propulsores de la aproximación a la Web 2.0 creen que el uso de la Web está orientado a la interacción y redes sociales, que pueden servir contenido que explota los efectos de las redes creando o no webs interactivas y visuales. Es decir, los sitios Web 2.0 actúan más como puntos de encuentro, o webs dependientes de usuarios, que como webs tradicionales. [16].

En 2005, Tim O'Reilly definió el concepto de Web 2.0.

El termino Web 2.0 hace referencia a un conjunto de aplicaciones usadas en aplicaciones de Internet que utilizan la sinergia de cada una de estas aplicaciones para un objetivo único, darle al usuario el control de sus datos.

De esta manera, podemos entender a las aplicaciones Web 2.0 como:

"todas aquellas utilidades y servicios de Internet que se sustentan en una base de datos, la cual puede ser modificada por los usuarios del servicio, ya sea en su contenido (añadiendo, cambiando o borrando información o asociando metadatos a la información existente), bien en la forma de presentarlos, o en contenido y forma simultáneamente." [10]

La infraestructura de la Web 2.0 es compleja y va evolucionando, pero incluye el software de servidor, sindicación de contenidos (RSS), protocolos de mensajes, navegadores basados en estándares, y varias aplicaciones para clientes.

Entre estas técnicas podemos destacar:

- CSS, marcado XHTML válido semánticamente y Microformatos
- Técnicas de aplicaciones ricas no intrusivas (como AJAX)
- Java Web Start
- XUL
- Sindicación/Agregación de datos en RSS/ATOM
- Urls sencillas y con significado (SEM)
- Soporte para postear en un blog
- XML
- Algunos aspectos de redes sociales
- Mashup (aplicación Web híbrida)

Una aplicación Web 2.0 tiene como característica principal, la facilidad para controlar los datos de la misma, ya sea para extraer como para poder introducir datos con facilidad, a su vez los usuarios son los únicos de dueños de sus datos y los pueden manejar a su manera pero de forma controlada desde cualquier navegador basado en estándares.

Redifusión de contenido

Este concepto utiliza protocolos estándares que les permite a los usuarios poder ver los contenidos de la aplicación Web desde otros contextos ya sea en otra Web, o una aplicación de escritorio, este los protocolos estándares de redistribución se encuentran RSS, RDF (conocido también como RSS 1.1), y Atom, todos ellos variedades de XML.

Servicios Web

Los protocolos de mensajes bidireccionales son uno de los elementos clave de la infraestructura de la Web 2.0. Los dos tipos más importantes son los métodos RESTful y SOAP. REST indican un tipo de llamada a un servicio Web donde el cliente transfiere el estado de todas las transacciones. SOAP y otros métodos similares dependen del servidor para retener la información de estado. En ambos casos, el servicio es llamado desde un API. A veces este API está personalizado en función de las necesidades específicas del sitio Web, pero los APIs de los servicios Web estándares (como por ejemplo escribir en un blog) están también muy extendidos. Generalmente el lenguaje común de estos servicios Web es el XML, si bien puede haber excepciones.

Recientemente, una forma híbrida conocida como Ajax ha evolucionado para mejorar la experiencia del usuario en las aplicaciones Web basadas en el navegador. Esto puede ser usado en webs propietarias (como en Google Maps) o en formas abiertas utilizando un API de servicios Web.

Software de servidor

La funcionalidad de Web 2.0 se basa en la arquitectura existente de servidor Web pero con un énfasis mayor en el software dorsal. La sindicación sólo se diferencia nominalmente de los métodos de publicación de la gestión dinámica de contenido, pero los servicios Web requieren normalmente un soporte de bases de datos y flujo de trabajo mucho más robusto y llegan a parecerse mucho a la funcionalidad de intranet tradicional de un servidor de aplicaciones. El enfoque empleado hasta ahora por los fabricantes suele ser bien un enfoque de servidor universal, el cual agrupa la mayor parte de la funcionalidad necesaria en una única plataforma de servidor, o bien un enfoque plugin de servidor Web con herramientas de publicación tradicionales mejoradas con interfaces API y otras herramientas. Independientemente del enfoque elegido, no se espera que el camino evolutivo hacia la Web 2.0 se vea alterado de forma importante por estas opciones.

4. Aplicaciones Ricas de Internet

Las Aplicaciones Ricas de Internet son la nueva generación de aplicaciones Web cuyo principal foco de atención es el usuario, es un nuevo tipo de aplicación con más ventajas que las tradicionales aplicaciones Web, entre ellas podemos destacar como se trabajo en mejorar la arquitectura de las mismas para lograr efectos eficientes y acortar la brecha entre las aplicaciones Web y las de escritorio.

Una aplicación Rica en Internet es enteramente un nuevo grupo de experiencias Web que involucra interacción, velocidad y flexibilidad.
En los entornos RIA no se producen recargas de página, ya que desde el principio se carga toda la pagina y los datos necesarios para que aplicación funcione, y podríamos discutir si cargamos o no datos que a un futuro no muy lejano estaremos necesitando.
Sólo se produce comunicación con el servidor solo cuando los datos son requeridos (on-demand), cuando se necesitan datos externos como datos de una base de datos o de otros ficheros externos.

En la actualidad hay muchas empresas que están haciendo hincapié en el desarrollo de herramientas para crear aplicaciones Web ricas, entre estas podemos mencionar las plataformas como:

- Flash y Flex de Adobe,
- AJAX
- Open Laszlo (herramienta Open Source)
- SilverLight de Microsoft
- JavaFx Script de Sun Microsystems

Las Aplicaciones Ricas en Internet combinan los mejor de los estándares, tecnologías, prácticas, para llevar a cabo su principal objetivo, mejorar la calidad de las aplicaciones Web.
Son aplicaciones que abarcan amplio espectro de áreas y bajo costo de desarrollo, además de esto provee lo mejor de la interactividad y con muchas propiedades de multimedia disponibles.
Cuando mezclamos – coordinamos todas estas cosas casi automáticamente estamos teniendo aplicaciones más “ricas”, dándole al usuario aplicaciones con las siguientes características:

- mas intuitivas
- mas responsables
- mas efectivas
- mas parecidas a las aplicaciones de escritorio
- mas interactivas
- mas dinámicas
- mas rápidas
- sin refrescos de pagina
- desarrollo casi instantáneo

- multi - plataforma,
- descargas progresivas para recuperar contenidos y datos,
- adopta ampliamente los estándares de Internet
- diseño con múltiples capas

Si hacemos un repaso sobre la lista de características que acabamos de nombrar, nos damos cuenta que satisfacen la mayoría de los objetivos que deseamos de las aplicaciones de hace una par de años. Desde el punto de vista del desarrollador la mayoría de este aspecto son tenidos en cuenta, pero hasta la actualidad no existían la cantidad de herramientas que existen hoy en día para poder desarrollar una aplicación a conciencia.

En las aplicaciones Ricas en Internet, el cliente empieza a tener mayores responsabilidades, ya deja ser solo un simple dibujante de la pagina, en esta nueva etapa de las aplicaciones Web el cliente pasa a tener responsabilidad en la forma en que se piden los datos al servidor, a partir de ahora hace mas explícitamente el envío y recepción de datos en modo asincrónico de un requerimiento de usuario, redibujar las regiones de la pagina, reproducir audio y video los cuales están estrechamente relacionados, independientemente del servidor o back end al que se encuentre conectado.

La arquitectura de las aplicaciones Ricas en Internet asigna esta responsabilidad al cliente para poder acercarse más a las aplicaciones de escritorio y dejar de ser un simple cliente en una arquitectura cliente/servidor.
Esto se logra agregando funcionalidad directamente sobre la arquitectura de las aplicaciones de n-capas que utilizan las aplicaciones legacy sin necesidad de tener que revisar el resto de las capas.

Esto puede ayudar a solucionar la mayoría de los problemas antes mencionados, como así también permite desarrollar aplicaciones con requerimientos complejos reduciendo el costo de desarrollo así como también permitiendo el desarrollo de la misma en primer lugar.

Las características de flash hace que un en día la mayoría de las aplicaciones Ricas en Internet lo usen como un cliente ligero y de bajo uso de ancho de banda en comparación con las aplicaciones actuales de escritorio, uno de los principales beneficios de las aplicaciones RIA que utilizan Flash es que están soportados por una amplia gama de plataformas y dispositivos, ya que funciona en distintos tipos de sistemas operativos y diferentes browsers. Pero Flash no deja de ser una herramienta difícil de manejar, y poco abierta a realizar modificaciones, lo que hace que los desarrolladores usen flash como una segunda alternativa.

Las aplicaciones RIA cobran su potencia a través de su arquitectura y capacidades para fundamentalmente cambiar la forma en que las compañías se comprometen e interactúan con los usuarios Web, dándoles mejores experiencias de usuario con resultados sobresalientes.

4.1. Que Necesitamos de las aplicaciones Web

La mayor demanda de aplicaciones Web es relativa a aplicaciones de negocios, esta claro que la demanda de este tipo de aplicaciones esta en escalada, ya sea aplicaciones para clientes y proveedores como aplicaciones de intranet.

Cuando la Web apareció las tecnologías clientes no tenían carga funcional, esto resulto en un punto ultra negativo, y hacen de las aplicaciones Web, sean pobres, con falta de interacción con el usuario resultando en malas experiencias

Hoy en día las aplicaciones Web deben tener en cuenta otros aspectos al momento de ponerse en marcha.

Para solucionar los problemas diarios de las aplicaciones Web deberán:

- utilizar un cliente activo
- Correr indistintamente Internet en las diferentes plataformas
- Que las aplicaciones funcionen correctamente tanto con conexiones de gran ancho de banda como conexiones de bajo rendimiento
- Mover el poder de procesamiento hacia el cliente (no solo de renderización)
- Usar interfaces de usuario con alto grado de interacción
- Representar procesos, configuración de datos, la escala y complejidad de procesos
- Utilizar audio, video imágenes y texto sin que se entrecorte
- Permitir al usuario trabajar tanto online como offline
- Permitirle al cliente determinar que contenidos o datos acceder y cuando (recuperación de contenidos de manera asíncrona)
- Acceso a múltiples servicios de nivel intermedio y datos almacenados de back-end
- Proveerle al usuario un potente front-end que involucre Web Services usando estándares tales como XML y/o SOAP
- Integrar con aplicaciones y sistemas legacy
- Permitir una incorporación gradual de nuevas funciones a las que ya existen en la actual aplicación y así poder sacar el mayor provecho de las inversiones existentes de aplicaciones Web.

Las tecnologías candidatas para estas soluciones no simplemente deberán ocuparse de las limitaciones del modelo, sino que también deberán proveer la capacidad para crear nuevas formas de participación y Aplicaciones innovadoras.

4.2. Consideraciones

La Web es un canal muy importante en las relaciones entre clientes y compañías, sin embargo las compañías ejecutivas a menudo manifiestan su descontento por no haber llegado a poner en producción en Internet una aplicación que este a la altura de crecimientos y las posibilidades que ofrece la Internet hoy en día.

En la actualidad es muy amplio el espectro de aplicaciones Web sobre la red, Las aplicaciones ricas en Internet pueden ser aplicadas a cualquiera de los diferentes rubros.
Una de las principales virtudes es que le permite reducir la complejidad de la escalabilidad de las aplicaciones Web actuales.
Las aplicaciones ricas son un puente entre las compañías y la visión de las aplicaciones Web, de esta manera las compañías están más cerca de los clientes y del impacto de las aplicaciones Web sobre los negocios en Internet

4.3. **Nuevas características**

- Desarrollar aplicaciones con nuevas características y capacidad que se harían muy dificultosas o casi imposibles de desarrollar en las aplicaciones Web tradicionales

- Mejorar la interacción con el cliente, devolver resultados en línea mas inmediatamente, mas leales , mejorando de esta manera el servicio que la aplicaciones puede brindar

- Crear sitios Web atractivos usando audio, video texto , gráficos que generan clientes potenciales, incrementar las ventas y simplificar la comunicación

- Simplifica los procesos tales como registración, configuración o compras, ayudando a incrementar las ventas, archivos y tiempo en el sitio además de la repetición de las visitas

- Reduce los costes del ancho de banda asociados con los refrescos de la página, que producen alto tráfico.

- Incrementar de manera considerable las ventas de sus productos y servicios a través de Internet

- Reducir los costos de desarrollo comparado si usase una tecnología alternativa para lograr los efectos de RIA

4.4. Ventajas

Las aplicaciones ricas tienen la capacidad de correr en la mayoría de los navegadores, dados los estándares que adopta, pero desarrollar una aplicación que corren en un browser las hace mucho más limitadas, de todas maneras asumir los esfuerzos de desarrollo tiene sus frutos, a continuación se detallan las ventajas:

- No requieren instalación – actualizar y distribuir la aplicación resulta instantáneo.

- Las Actualizaciones a nuevas versiones son automáticas

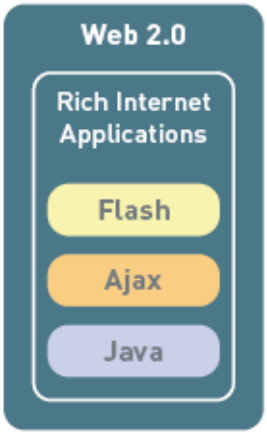
- Los usuarios pueden usar la aplicación desde una computadora con conexión a Internet, y usualmente si interesar que sistema operativo se encuentra instalado en la computadora.

- La aplicaciones Web son generalmente menos propensas a la infección de un virus que una aplicación que corre en un ejecutable porque las aplicaciones RIA manejan las interacciones con el usuario a través de un motor cliente

- Aplicaciones más "*ricas*": Podemos darle comportamiento a las interfaces de usuario cosa que antes era imposible lograrlo con widgets HTML disponibles para los browser más comunes de las aplicaciones Web. El enriquecimiento de funcionalidades puede incluir alguna cosa que pueda ser implementada en la tecnología para ser usada del lado del cliente, como por ejemplo Drag & Drop, usar un componente deslizante que me permita cambiar los ítems, ejecutar cálculos del lado del cliente sin necesidad de enviar un request al servidor, por ejemplo una calculadora.

- Aplicaciones más ágiles: los comportamientos de interfaces típicamente son mucho más ágiles que los que proveían las aplicaciones Web estándares a través de las sucesivas interacciones con el servidor.

4.5. Arquitectura de una aplicación RIA



La figura muestra la relación entre Web 2.0, RIA, Ajax y Flash

Los elementos que muestra la figura son los principales encargados del comportamiento, la performance y el manejo de las aplicaciones RIA.

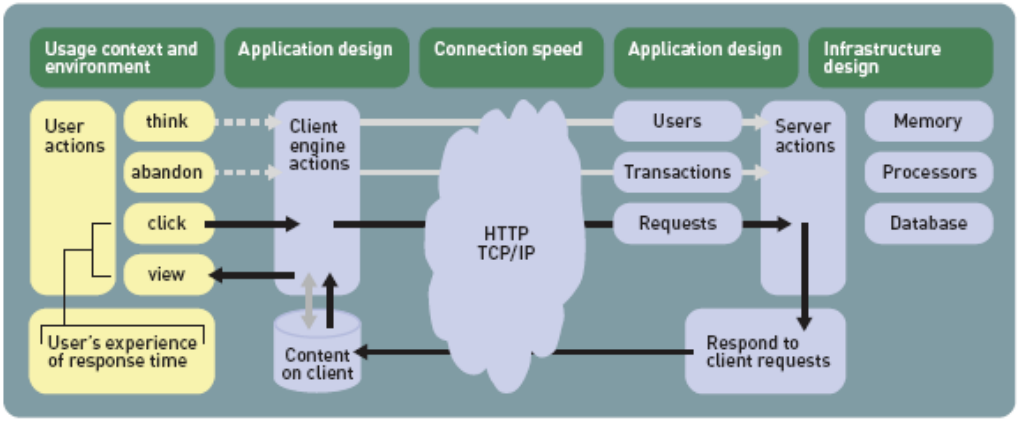
Hay tres aspectos muy importantes que influyen en forma directa en la performance de una aplicación RIA.

- *El diseño de la aplicación y el contexto o ambiente de uso*
- *El comportamiento y expectativas del usuario*
- *El comportamiento de la aplicación cuando la estamos utilizando*

Desde que el usuario inicia una petición a una dirección URL hasta que la respuesta es enviada al usuario, se suceden una serie de pasos los cuales intentaremos explicar en los siguientes ítems

4.5.1. Interacción entre el cliente y el servidor

El usuario hace un click en un link del browser envía un request a un Server. El servidor responde a los request de los clientes, y cuando la cantidad suficiente de los contenidos requeridos llegan al cliente (a la cache del browser), el browser los muestra y el usuario los puede ver. Para que el usuario pueda ver la respuesta deberá esperar el tiempo necesario que tarde el proceso completo en que la respuesta llegue al browser.
La fecha negra del diagrama muestra el flujo de descarga de una página Web tradicional



Una pagina Web tradicional generalmente esta compuesta por hojas de estilo, archivos de scripts, imágenes, objetos flash entre otros, todos estos componentes son descargados, individualmente, varias veces ante los sucesivos request ya que la mayoría de las veces requiere varios intercambios de información entre el servidor y el cliente.

4.5.2. Comportamiento del servidor

Los servidores deben dar respuesta a muchos usuarios concurrentemente, no interesa cuan poderoso es el servidor, cada usuario que hace un request al servidor consume un pequeña parte de recursos del mismo, entre los cuales podemos destacar:

- Memoria
- ciclos de procesador
- recursos de motores de base de datos.

Los servidores Web pueden responder rápidamente a requerimientos de información de usuarios concurrentes, creando caches de browseo haciendo mucho más eficiente las respuestas del mismo.
Pero una acción de usuario que involucra cambios en los datos tales como agregar un producto al carrito de compras, consume mas recursos de servidor.
El número de transacciones concurrentes, interacciones que actualizan los datos personales de un cliente, juegan un rol crítico en la performance del servidor.

Las flechas grises del diagrama y los recuadros de usuarios y transacciones indican que la performance del servidor es altamente influenciada por estos factores concurrentes.
Los servidores típicamente funcionan de manera coherente hasta un cierto nivel de concurrencia, pero más allá de ese nivel (punto de inflexión), la performance de las transacciones se ven degradadas, transformadose en un cuello de botella.

Como resultado tenemos que con pequeños cambios (refactorings) podemos mejorar los tiempos de respuesta significativamente transformado en un mejor visión del usuario para con la aplicación.

El flujo del envío y recepción de datos de una aplicación rica es marcado en la figura anterior con líneas punteadas.
En la siguiente figura podemos ver las diferencias de carga, performance y consumo de recursos entre una aplicación rica y una aplicación convencional

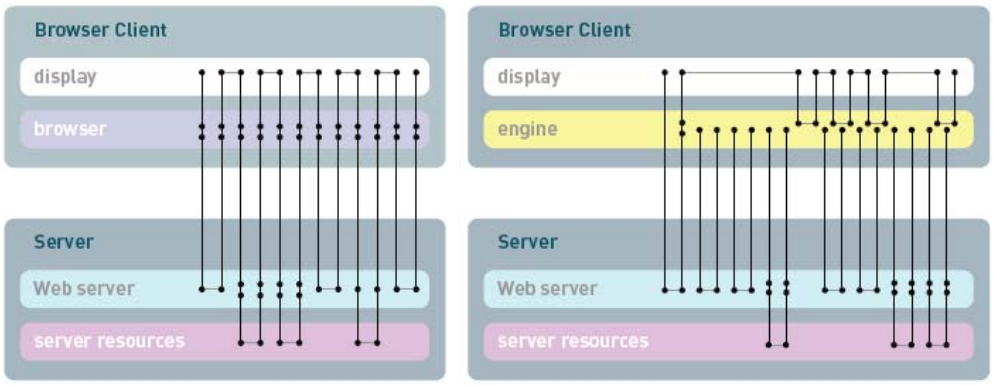


Figura 3ª (izquierda). Modelo de comunicación de aplicación Web tradicional
Figura 3b (Derecha). Modelo de comunicación de un modelo de aplicación RIA

Hay un factor importantísimo que deberemos tener en cuanta en las aplicaciones ricas, cualquiera de los frameworks que se utilicen para desarrollar implícitamente proporciona un elemento mas a las capas de la aplicación, esa capa también puede ser llamada “Motor Cliente”, esta capa cliente la podemos ver en la figura anterior, en el recuadro amarillo.

4.5.3. El “Motor” Cliente

Hoy en día hay varias implementaciones diferentes, todas las RIAs agregan una capa lógica intermedia, el “motor cliente de pedidos”; Esta capa es agregada entre el usuario y el servidor Web. Este “motor” maneja que contenidos a mostrar y los request hacia el servidor.
Este “motor” cliente puede seguir manejando las comunicaciones sincronas como las aplicaciones Web tradicionales, pero también permite al usuario interactuar con la aplicación asincrónicamente (independientemente de forma en que nos comuniquemos con el servidor).
La figura 3ª y 3b muestran como es un patrón de comunicación asíncrona usuario-servidor en una aplicación RIA, y como difiere de una comunicación tradicional sincronía de una aplicación Web.

Dependiendo de la plataforma de desarrollo la implementación difiere, todas las aplicaciones RIA agregan una capa intermedia o lógica, este es el motor del

cliente encargado de gestionar los requerimientos de forma background, esta capa es ubicada lógicamente entre el usuario y el servidor Web.
Este “Motor” se descarga en el cliente al inicio de la session y maneja los cambios que se producen sobre la pantalla y se los comunica al servidor. Este motor solo produce cambios sobre los elementos necesarios sobre la pantalla, evitando las recargas completas de paginas.

4.6. Principales Características

4.6.1. Anticipar carga de datos

La información puede ser cargada de un servidor en un requerimiento anticipando el request del usuario, podemos preveer los datos que el usuario va a necesitar en request anterior, muchas veces se da lo opuesto, solo cargamos datos necesarios para optimizar recursos

4.6.2. Evitar recarga completa

Muchas veces cambiamos el valor de un input que envía un evento de servidor, por ejemplo en .Net si el input tiene asociado un evento de servidor, la aplicación recarga por completo la página, en las aplicaciones RIA esto no ocurre, y en .Net podemos evitar esto utilizando componentes AJAX

4.6.3. Múltiple validación

La aplicación realiza la mayoría de las validaciones en la capa cliente de la aplicación, de esta manera evitamos enviar datos inválidos al servidor.

4.6.4. Respuestas sin intervención del servidor

Validando en el cliente automáticamente podemos enviarle al usuario una respuesta, sin necesidad de que sea el servidor el encargado de validar datos de formularios ingresados por el usuario.

4.6.5. Anticipar eventos

Así como podemos anticipar el envío de datos al cliente, podemos anticipar ejecución de eventos de servidor, de esta manera cuando ejecutamos un evento podemos anticiparnos y ejecutar otro que por definición de la aplicación se ejecuta tiempo después.

Los diseñadores pueden explotar estas posibilidades para lograr aplicaciones mas ágiles garantizando experiencias de usuario más receptivas.
Ciertamente, en la mayoría de los ejemplos, los usuarios gastarán menos tiempo en esperar una respuesta del servidor, sin embargo, al aplicar estas técnicas estaremos teniendo modelo de datos mucho más complejos que el de una aplicación Web tradicional.

4.7. Ventajas de la arquitectura

Las aplicaciones Ricas en Internet tienen la característica de exhibir un "look and feel" que se aproxima al de una aplicación de escritorio, Pero también tiene tiempos de respuesta parecidos a las mismas, dada la arquitectura que define.

Dentro de las principales características podemos destacar:

4.7.1. ***Balance de carga entre el cliente y el servidor***. La demanda de recursos de computación entre le cliente y el servidor es mas balanceada, tal es así que los servidores Web no necesitan ser grandes servidores como se necesitaban en las aplicaciones Web tradicionales. Estos recursos libres que quedan disponibles permiten a un mismo servidor poder manejar mas sesiones de usuarios concurrentemente.

4.7.2. ***Comunicación asíncrona***. El motor cliente puede interactuar con el servidor sin tener que esperar para que el usuario puede completar una acción por un click en la interfaz, sobre un botón o bien sobre un link. Esto le permite al usuario ver e interactuar con la página *asincrónicamente* desde el motor cliente comunicándose con el servidor. Esta opción permite a los diseñadores RIA mover datos entre el cliente y el servidor sin hacer esperar al usuario como ocurría antiguamente con las aplicaciones comunes. La aplicación mas común de esto es la obtención previa , en el cual una aplicación anticipa la obtención de cierto dato y descarga esto al cliente antes de que el usuario lo requiera, esto es acelerado en una respuesta subsecuente. *Google maps* usa esta técnica para moverse en un mapa antes de que el usuario escrolee.

4.7.3. *Eficiencia en el tráfico de red.*

El tráfico de red puede ser reducido significantemente porque el motor de una aplicación puede ser más inteligente que un cliente común cuando decide que datos son necesarios en un intercambio con el servidor. Esto puede acelerar un requerimiento individual o una respuesta porque son menos los datos que se intercambian en cada interacción, y sobretodo reduce la carga de la red. Sin embargo el uso de la técnica de *prefeching* puede neutralizar o jugarle en contra en cuanto al trafico de red porque el código no puede anticipar con certeza que es lo que el usuario necesitara anticipadamente, esto es muy

común para tales técnicas de descarga de datos extra ya que no todos serán necesitados para algunos o todos los clientes. [22]

4.8. Herramientas de diseño RIA

Las aplicaciones Ricas en Internet todavía se encuentran en una etapa de desarrollo, y esperando la opinión de los usuarios.

A continuación detallamos un conjunto de restricciones y requerimientos a tener en cuanta que las herramientas de desarrollo deberán contemplar:

Browser Modernos: Las aplicaciones Web por lo general requieren browser modernos para poder funcionar correctamente, además de esto será necesario un motor javascript que sea el encargado de manejar requerimientos XMLHTTPRequest para manejar las comunicación del cliente con el servidor y viceversa; y el principal condimento es que deberán adoptar los estándares DOM y manejo de hojas de estilo CSS

Estándares Web: uno de los principales problemas de escribir aplicaciones con funcionalidad javascript, es que todos los browsers no adoptan los mismos estándares y eso hace que debamos escribir líneas de código adicional para que funcionen en los diferente browsers

Herramientas de Desarrollo: Algunos frameworks AJAX y productos de desarrollo como Curl, Adobe Flex and Microsoft SilverLight proveen un entorno de desarrollo integrado para crear aplicaciones Web ricas

4.8.1. Javascript / Ajax

El mejor lenguaje la tecnología cliente disponible con la habilidad de correr código e instalado en la mayoría de los browser es javascript.

Aunque el uso del lenguaje estaba relativamente limitado, cuando combinamos con el uso de capas en las aplicaciones Web y el uso de de otros desarrollos en DHTML comenzó a ser una alternativa cada vez mas frecuente para la construcción de aplicaciones ricas en Internet.
Ajax es un término nuevo acuñado para referirse a esta combinación de técnicas y recientemente ha sido mas usado por Google para los proyectos como Gmail y Google maps
Sin embargo, crear una aplicación grande con este Framework es muy difícil, deben interactuar muchas tecnologías diferentes y tener buenos resultados, a su vez que sea compatible con el navegador requiere una gran cantidad de esfuerzo
Para simplificar el proceso, varios Frameworks open source han sido desarrollados, así como también Frameworks comerciales.

4.8.2. Google's GWT Framework

Google fue uno de los pioneros en lanzar al mercado allá por el 2006 una herramienta para el desarrollo de aplicaciones ricas en Internet, el 'Google Web Toolkit' también llamado GWT.
Este es un Framework que nos permite desarrollar y testear aplicaciones ricas en Internet utilizando el lenguaje de programación JAVA.
El paradigma de programación de GWT se centra en codificar la lógica de la interfaz de usuario en Java (similar al modelo de programación de Swing/AWT), y ejecuta el compilador de GWT para traducir a una lógica que pueda correr en los browsers compatibles con javascript.
Este Framework fue desarrollado específicamente para los programadores java, este me permite la programación, debug, testeo de aplicaciones ricas en Internet y además nos permite hacer refactorings, todo esto podemos lograrlo Utilizando herramientas conocidas como puede se Eclipse y aunque suene asombroso podemos hacerlo sin la necesidad de tener conocimientos de programación en javascript o el conocimiento de el DOM de los browser's. [21].

4.8.3. Adobe Air

Es una tecnología que aprovecha las mismas tecnologías que se usan en la red (HTML, AJAX, Flash, Flex,..) para desarrollar aplicaciones para el escritorio. Entre las principales características podemos destacar que es multiplataforma, una misma aplicación servirá tanto para Windows como para Mac OS X y aunque un poco más tarde también para Linux. Se basará en Webkit, el motor de renderizado con el que funcionan navegadores como Konqueror y Safari, además de en las otras tecnologías de Adobe Flash y Flex, pero no se queda solo en eso sino que ofrecerá una serie de APIs para facilitar el desarrollo de aplicaciones para el escritorio (se estima que a principios de 2008 se lance la primera versión final). Entre ellas encontramos la posibilidad de acceder a ficheros locales, una base de datos
SQLite, soporte de drag & drop, acceso al portapapeles y, en general, soporte para que la aplicación se instale como un programa en local, con su instalación, sus iconos, etc. [1]

4.8.4. Microsoft Silverligth

Microsoft ha presentado SilverLight 1.0, en sus versiones para Windows y para Mac, y pronto estará disponible para Linux con el nombre de MoonLigth. Es otra de las tecnologías que quiere apostar por una experiencia más interactiva en Internet. Es un conjunto de tecnologías multiplataforma que permite trabajar con animaciones, gráficos vectoriales y vídeo, así como la carga de datos desde el escritorio del usuario. Una de las características más destacadas es que permite mostrar vídeo en alta calidad.
Microsoft SilverLigth utiliza la tecnología WPE que permite la creación de experiencias interactivas, ricas y visualmente sorprendentes que se pueden ejecutar en cualquier lugar. WPE puede interactuar con la tecnología XAML mediante JavaScript [13].

4.8.5. Appcelerator

Appcelerator es una plataforma open source para le desarrollo de aplicaciones ricas en Internet, utiliza un servicio orientado a la arquitectura y estándares tales como HTML, CSS y javascript.
Las aplicaciones Appcelerator pueden ser integradas automáticamente en diferentes capas de servicios desarrolladas con Java, PHP, Python, .NET y Perl.
A su vez también pueden usarse para crear widgets y engancharlos con las aplicaciones ricas en Internet.

4.8.6. OpenLaszlo

OpenLaszlo es un Framework de desarrollo para aplicaciones ricas en Internet open source creado por Laszlo Systems Inc.
El compilador OpenLaszlo compila los programas escritos en LZx Languaje (que son mix de tags xml y javascript) dentro de cualquier DHTML (comúnmente conocido como AJAX) o Adobe Flash bytecod, los cuales soportan flash7 y flash8.

4.8.7. MagicAjax.NET

MagicAjax es un Framework open-source, diseñado para facilitarle la tarea de desarrollo a los programadores de aplicaciones Web, muy fácil de usar, permite agregarle AJAX a las nuestras paginas Web de una manera muy intuitiva, con esta herramienta podemos seguir usando nuestros controles ASP.NET, y no necesitamos escribir código javascript de mas. [6].

4.8.8. Curl 5.0, REBOL 2.6 y Seaside for Smalltalk

Las alternativas disponibles Aplicaciones Ricas en Java incluyen maquinas virtuales para Curl, Rebol y smalltalk.
Curl no facilita la persistencia de datos en la capa cliente, REBOL no requiere un browser y smalltalk utiliza una version reducida del lenguaje para proveer una experiencia web mucho mas rica.

4.8.9. JavaFX

Sun Microsystems anuncio JavaFX, es una familia de productos basada en la tecnología JAVA, diseñado para proveer experiencias consistentes para una variedad de dispositivos incluyendo aplicaciones de escritorio como pueden ser applets y clientes que pueden funcionar por si solos.
La plataforma JavaFX inicialmente consistirá de JavaFX Script y JavaFX Mobile.

JavaFX Script permite el rápido desarrollo de interfaces ricas 2D usando una sintaxis similar a SGV (Scalable Vector Graphics).
Sun planea sacar al mercado a JavaFX Script como un proyecto open source, pero JavaFX Mobile será un producto comercial disponible a través de licencias para transportistas y los fabricantes de telefonía celular.

4.8.10. Java

Las aplicaciones ricas en Internet escritas en java pueden ser ejecutadas en un navegador como si fuera una aplicación común.
Las Aplicaciones Ricas en Internet basadas en Java tienen la ventaja de usar todo el potencial de la plataforma Java para darle riqueza a las aplicaciones ya sea con gráficos 2D o 3D.
Java es una plataforma que adopta un amplio rango de librerías comerciales y abiertas disponibles para la plataforma, posibilitando incluir soporte para cualquier sistema, incluyendo aplicaciones nativas por JNI o JNA
En la actualidad existen números frameworks para el desarrollo de aplicaciones Ricas en Internet con Java, entre ellos podemos nombrar XUI, Swixml, Canoo's y UltraLightClient.

4.8.11. Controles ActiveX

Usar controles dentro de HTML es una manera muy eficaz para desarrollar aplicaciones ricas en Internet, sin embargo las desventaja mas importante de esta técnica es que su correcto funcionamiento solo esta garantizado en Internet Explorer, ningún otro navegador de soporta controles ActiveX.
Hay que tener en cuenta que hay mus virus y malware que se propagan bajo esta técnica, lo que hace que si habilitamos este tipo de controles estaremos bajo la mira males potenciales.
Usar esta técnica es una buena alternativa solo si las empresas tienen adoptado Internet Explorer como navegador por defecto, sino lo ideal será buscar otra alternativa.

4.8.12. Estándares RIA

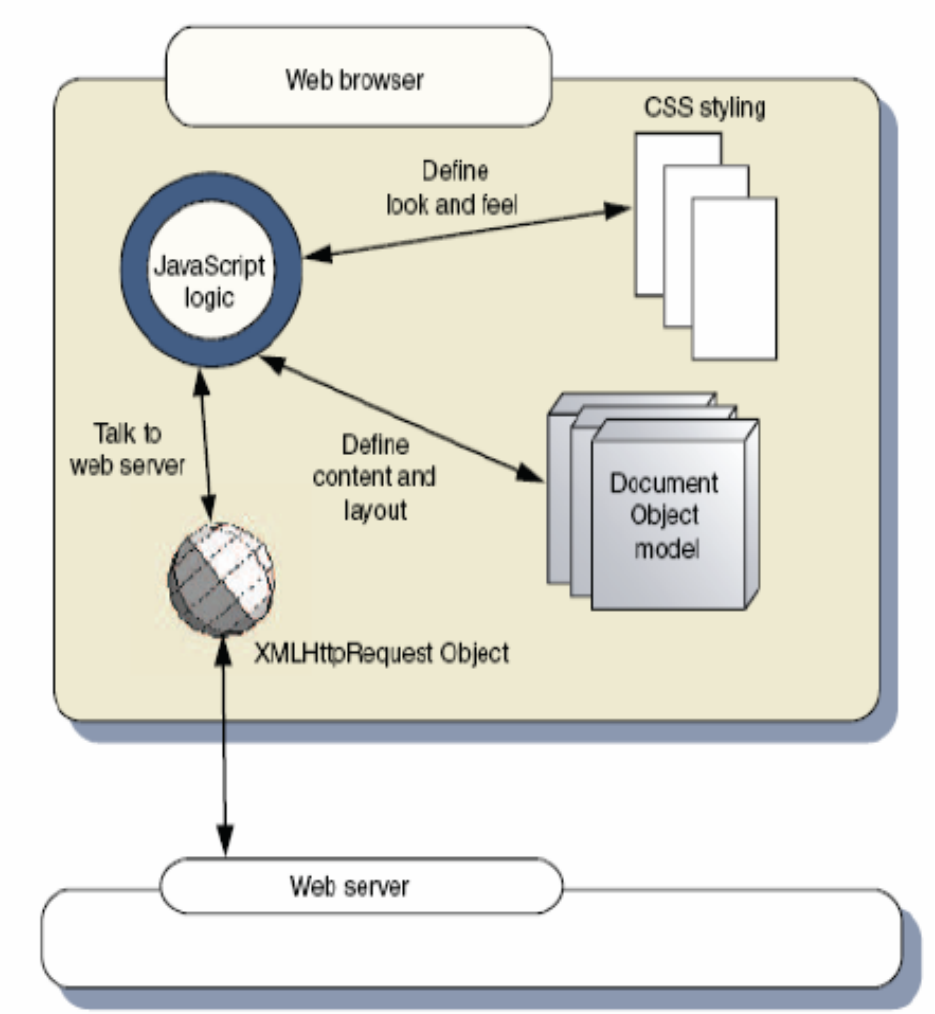
Usando HTML/XHTML, nuevos lenguajes basado, inspirados en estos podrían ser usados en aplicaciones Ricas en Internet
Por ejemplo, La Fundación Mozilla desarrollo un lenguaje de etiquetas de interfaz de usuario basado en XML, que podría ser usada en aplicaciones ricas en Internet pero estaría restringido para navegadores Mozilla, lo que hace que no sea un lenguaje estándar, lo que hace que nos salgamos de nuestra idea original.
La Actividad de Rich Web Clients ha iniciado un grupo de Trabajo para el estudio de Formatos Web cuya misión incluye el desarrollo de estándares para la tecnología.

El instituto tecnológico de masachuzet es uno de los principales involucrados en el desarrollo del W3C también desarrollaron un lenguaje de contenidos Web CURL el ya esta por la versión 5.
Las interfaces de usuario de RIA también se han empezado a enriquecer a través del uso de vectores gráficos escalables a través de scripts así como Synchronized Multimedia Integration Language (SMIL), esta funcionalidad no todos lo browser la soportan.

5. AJAX

Es una combinación de tecnologías que se desarrollan independientemente:

- XHTML y CSS: para crear una presentación basada en estándares
- DOM para la interacción y manipulación dinámica de la presentación,
- XML, XSLT y JSON para el intercambio y la manipulación de información
- XMLHttpRequest para el intercambio asíncrono de información y JavaScript para unir todas las demás tecnologías.



Modelo de una aplicación Que utiliza AJAX

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano. Las peticiones HTTP al servidor se transforman en peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren

intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere la respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

AJAX no es una nueva tecnología ni un producto, es una herramienta poderosa para el desarrollo de RIA que se basa exclusivamente en JavaScript, XML, entre otras.

Los productos JavaScript AJAX son muy útiles en situaciones donde el usuario espera que una aplicación funcione incondicionalmente del navegador sobre el cual esta corriendo.
Cuando compramos un DVD o queremos reservar un vuelo, por lo general no queremos que nos aparezca un cartel que no pida que instalemos un plug-in.
El HTML simple continúa siendo la mejor solución para requisitos simples de la interfaz de usuario A pesar de eso, un lugar que vende DVD's o bien las aplicaciones de reservación de viaje con interfaces de usuario simples a menudo son difíciles de usar o resultan inútiles, alguien que ha tratado de registrar un itinerario con vuelos múltiples, hoteles, y los autos de alquiler estarán de acuerdo que los sitios Web actuales son muy difíciles de usar.

La única desventaja de esto es que programar en javascript es más difícil,
Dos gigantes de la Internet como Amazon o Ebay no fomentan el potencial de JavaScript AJAX, aunque la tecnología ha estado aproximadamente por años y potencialmente mejoraría sus sitios Web sustancialmente
Simplemente supóngase cómo sus sitios se beneficiarían de una búsqueda que depende de cómo usted teclee, escroleo de largas listas, actualizaciones de pantalla parciales, o browsear árboles de productos categorizados

5.1. El rol de AJAX en las aplicaciones RIA

Los usuarios por lo general usan una aplicación Ajax tipeando en la URL del navegador, haciendo un click sobre un link, etc. Las aplicaciones que contiene Ajax son descargadas en el navegador y el usuario puede empezar a trabajar con ellas inmediatamente.

Ajax es un Framework cuyo principal foco es el usuario y sus necesidades,
Por demasiado tiempo la gente ha soportado de páginas que tardaban una eternidad en cargarse, controles difíciles de usar, Forms que cuando tardamos mucho tiempo y recargamos la página se pierden.
Sin embargo, Hay buenas razones para las personas continúen usando un navegador, esos son lo intereses de algunas compañías que se esfuerzan día a día para lograr aplicaciones interactivas, productivas y agradables.

AJAX apunta a mejorar la experiencia de usuario dentro de las restricciones de los estándares de los navegadores.

El diseño basado en estándares obviamente tiene los beneficios de que lo hacen portable, Sino que también tiene una usabilidad integrada dentro de la misma.
Los usuarios le interactúan con cientos de sitios Web, aprecian la consistencia, una calidad que rápidamente se pierde cuando usted confía en plugins, aplicaciones que solo funcionan en un navegador, e ideas "creativas" del diseño.
No es descabellado, agregar un nuevo control o capa con funcionalidad “rara” cuando trabajamos, pero para la mayoría, siempre es mejor trabajar con cosas estándares.

Las tecnologías que no son estándares, pueden romper con la consistencia, un ejemplo puede ser Flash, mucha gente lleva su control o aplicación de flash a AJAX porque lo usuarios no pueden descargar la imágenes haciendo click en el botón derecho del Mouse.
Concluyendo, Podemos usar flash para lograr cosas que con otros plugins no podríamos lograr, de hecho esa es la idea del mismo flash, pero no nos asegura la completa aceptación de la gente.

Algunas personas podrán decir que si la aplicación incluye flash no es una aplicación AJAX, lo cual no es totalmente cierto, ya que una aplicación podría delegar ciertas tareas flash o algún otro tipo de plugin.

Elegir un Framework Ajax no es lo mas importante de una aplicación hoy en día, la idea es preguntar que es lo que ayudaría mas a solucionar el inconveniente de los usuarios, algunas veces podría ser que ayude mas una aplicación programada con java, .Net o Flash, y otras una aplicación de escritorio.

Las tecnologías estándares, incluyen varias tecnologías bien definidas como pueden ser: HTML, CSS, HTTP, CGI, envío de formularios, scripting de código en el servidor.
Javascript, el Document Object Model (DOM), el modelo de eventos (DHTML), remoting Web, han comenzado a cobrar gran importancia a partir de su estandarización en los distintos navegadores
Hoy en día con la integración de tecnologías y las diferentes herramientas que utilizan este concepto para proveer componentes que no produzcan recarga de pagina y optimicen los recursos de las aplicaciones, se nos ve facilitada la tarea de desarrollo de aplicaciones con funcionalidades mucho mas rica y confiables para el usuario, que antes era casi imposible implementar sobre una aplicación Web.

6. Evolución de una aplicación Web tradicional a RIA

Migrar una aplicación por completo no es una tarea sencilla, necesitamos un plan de desarrollo, planificar tiempos de análisis, tiempo de diseño, tiempos de programación, etc.
Puede ser llegar a ser una piedra en el zapato para cualquier empresa, sobre todo si la base de la aplicación no tiene un buen diseño.

Un opción podría ser realizar un proceso de migración por fases, eso, mostraría al usuario el continuo compromiso de la empresa por mejorar la calidad de la aplicación y brindarle al usuario una mejor aplicación cada día.

Existe un concepto denominado “Refactoring”, un concepto poco conocido, poco nombrado, pero muy potente para poder mejorar la calidad de las aplicaciones con pequeños cambios sin producir grandes alteraciones al resultado final.

6.1. Refactoring

Según fowler un refactoring es una transformación que preserva funcionalidad, como Martin Fowler lo define, “un cambio en la estructura interna de un software para que sea mas fácil entenderlo y fácil de modificar sin cambiar el comportamiento del mismo” [5]

Muchas veces estamos en desacuerdo o no nos gusta como quedo escrita una clase, un método, una relación entre clases, una jerarquía, etc.
Es muy probable que el resultado que de la acción, que implica cualquiera de los inconvenientes antes mencionado, no se ve a afectado, pero podríamos mejorar o solucionar esto sin necesidad de cambiar la el resultado final.
A este proceso de mejorar alguno aspecto pequeño, sin cambiar la funcionalidad, se le llama refactoring.

Una cuestión muy importante que deberemos tener en cuenta al momento de hacer un cambio, es la inminente posibilidad de introducción de errores que se podrían dar, ya que es alta la probabilidad, así el cambio sea muy pequeño, debemos tener cuidado y testear bien la aplicación con los cambios introducidos.

Refactorizar en pequeños pasos previene este tipo de problemas, de esta manera realizar un cambio no lleva grandes cantidades de tiempo, algunos refactoring que involucren un volumen mas grande de cambios pueden llevar un poco mas de tiempo dependiendo de la complejidad del cambio, a su vez estos refactoring mas largos podrían dividirse en sub-refactorings para simplificar la tarea.

En un proceso refactoring, lo mejor es realizarlo en fases, introducir un refactoring, ponerlo en producción, testear los resultados, si hacemos muchos

cambios así sean pequeños, podemos perder el hilo o la posibilidad de detectar el origen, ya que deberemos testeas diferentes puntos de introducción de los mismos.

6.2. Ejemplos de Refactoring

En la actualidad existen una lista extensa de refactorings, para refactorizar aplicaciones orientadas a objetos, introduciremos el concepto y ejemplificaremos algunos para dar un pantallazo general del termino.
Hay varios tipos de refactorings; hemos escogido unos, de entre los más comunes, para examinarlos. Los nombres de los ejemplos corresponden a refactorings específicos

Para cada ejemplo, facilitaremos descripciones, situaciones, motivaciones y beneficios para ejecutar operaciones eficientes de refactoring.

6.2.1. Mover un Método

Motivación:
Un método se usa y/o es usado por más elementos diferentes de la clase a la que pertenece. Un fabricante de software asigna a cada desarrollador un ordenador.

Un método getComputerInsuredValue() ha sido añadido a la clase Developer, que va sumando o añadiendo valores de los componentes del ordenador en la clase Computer

¿Era la clase correcta donde añadir el método?

Queremos reducir o eliminar excesivos acoplamientos a nivel inferior entre clases. Si una clase accede directamente a los atributos de otra clase, eso puede impedir desarrollos futuros. Por ejemplo, si se cambia el tipo de un campo en la clase Computer, eso podría romper el método en la clase Developer. Un mejor diseño utilizaría métodos de acceso (get/set), lo que permitiría que los atributos permanecieran con ámbito privado.
Sin embargo, ¿hacen realmente falta métodos de acceso para los atributos? En lugar de crear métodos de acceso para los componentes del ordenador que permitan a la clase Developer leer sus valores, tal vez tendría mas sentido trasladar el método que es tan dependiente de esos atributos, a la misma clase que ellos. Eso evitaría la necesidad de un acceso público a dichos atributos y/o métodos de acceso.

Nuestro objetivo entonces, no es solo identificar el uso de atributos públicos de diferentes clases, sino también el de ubicar métodos en la clase apropiada.

Eso nos permitirá no solo reducir la necesidad de atributos públicos, sino también eliminar acoplamientos a nivel inferior entre clases. En este caso, cuatro operaciones de lectura separadas son reemplazadas con un método de alto nivel.

Mecanismo:

- El campo computer en la clase Developer class depende de la clase Computer.
- El método assignComputer() depende de la clase Computer.
- El método assignComputer()escribe un valor en el campo computer de Developer.
- El método getComputerInsuredValue()en la clase Developer lee los valores de cuatro campos diferentes en la clase Computer. Este último detalle lleva directamente a la idea de que se puede simplificar el código trasladando (o delegando) el método getComputerInsuredValue() a la clase Computer; un refactoring básico de tipo move method.

Moviendo el método getInsuredValue() desde la clase DEVELOPER a COMPUTER, de esta manera con un simple refactoring reducimos el grado de acoplamiento de nivel inferior

Impacto: Ubicar métodos en las clases apropiadas genera los siguientes beneficios

- Facilita futuros desarrollos, ya que la relación entre esas clases esta simplificada y mejor definida.

- Sustituir accesos múltiples de nivel inferior con un acceso de nivel superior puede mejorar el rendimiento incluso si las clases de este tipo acaban alguna vez distribuidas en diferentes máquinas.
  En este ejemplo, observaremos que las clases están demasiado acopladas y concluiremos que los métodos deberían moverse.

Consideraciones:
¿Cómo se pueden detectar todos los excesivos acoplamientos de bajo nivel entre clases?

¿Cómo se pueden resolver con seguridad los excesivos acoplamientos de bajo nivel?

¿Cómo se pueden recuperar todos los elementos de dichas relaciones de bajo nivel?

6.2.2. Extraer una Clase

Motivación:
Se creó una clase Developer y después se le fueron añadiendo atributos y métodos relacionados con el ordenador del desarrollador. Con el tiempo, se necesitó dividir la clase Developer y crear una clase Computer distinta.
Ósea que tenemos una clase hace dos cosas distintas o tiene dos áreas de responsabilidad diferentes.
Si una única clase se vuelve demasiado grande, puede ser un síntoma de un diseño regular al dar a la clase responsabilidades múltiples. Y a un nivel practico, las clases que son demasiado grandes dificultan el desarrollo.

Mecanismo:
Crear una nueva clase y mover los métodos y campos apropiados en la nueva clase.
En este caso lo ideal es hacer una distinción clara entre los métodos y campos relacionados con Computer (y que deben ser trasladados a la nueva clase) y los relacionados con Developer. Esto indica que, en este caso, se aconseja extraer una clase.
Con una clara visión de todo lo que necesita ser reemplazado, un programador puede entonces ejecutar el refactoring con confianza.

Impacto:
Después del refactoring, hay ahora dos clases distintas, cada una con un área de responsabilidad claramente definida. En particular, fíjense en el método assignComputer()en la clase Developer. Dicho método necesitaba anteriormente acceder a cada una de las partes del ordenador individualmente y ahora sencillamente establece el atributo computer de Developer. En otras palabras, un ordenador esta asociado con un desarrollador, salvo que los detalles del ordenador están encapsulados en la clase Computer, en lugar de llenar la clase Developer.
Tener clases pequeñas con áreas de responsabilidad e interfaces bien definidas brinda los beneficios siguientes:

- Se puede delegar el desarrollo a varios desarrolladores o equipos.
- Aumenta la posibilidad de reutilización.
- Facilita la comprensión del diseño de manera que los desarrolladores entiendan mejor el código con el resultado de una mayor productividad y una reducción del tiempo empleado en depuraciones y pruebas.
- Distribución más fácil.

Se ha mejorado el diseño.

Notas:
- ¿Cómo un desarrollador puede detectar las clases que se deben dividir?
- Cómo un jefe de proyecto puede valorar lo complejo que es el refactoring de implementar?
- La división se hará de manera limpia y natural o hará falta mucha lógica?

Lo ideal seria entender el comportamiento de la clase y que métodos y propiedades utiliza para cada proceso y que parte de su comportamiento forma ese proceso.

6.2.3. Eliminar intermediarios

Motivación: Un Manager necesita gran cantidad de información contenida en la máquina del desarrollador. Tal como fue diseñado originariamente, se utilizaban los métodos de conveniencia de la clase Developer para delegar las llamadas a los métodos en la clase Computer, esto es, una clase esta delegando demasiado.

Mecanismo
Para solucionar el problema lo mejor es hacer que la clase cliente llame directamente a la delegada.
La solución es dejar que la clase Manager llame directamente a la clase Computer, lo que puede necesitar modificaciones del código para exponer el ordenador del desarrollador a la clase Manager.
Un método de acceso getComputer() ha sido añadido a la clase Developer, lo que permite a la clase Manager obtener una referencia del ordenador del desarrollador. De ahí, la clase Manager puede interactuar directamente con la clase Computer, deshacerse de la clase intermediaria, simplificando el código y haciendo que los futuros desarrollos sean un poco más sencillos.

Impacto:
Si la delegación excesiva puede evitarse, los desarrolladores pueden entender mejor el diseño y el código se vuelve menos pesado.
En muchos casos, delegar es una buena elección de diseño, permite ocultar implementaciones y es una manera eficaz de agregar funcionalidades dispares en una interfaz común.
Sin embargo, en otros casos, delegar demasiado vuelve el código incómodo. Delegar excesivamente, puede enmascarar innecesariamente el acoplamiento real, lo cual sería conceptualmente mas claro si en su momento se expuso directamente. Y en algunos casos, solo crea código adicional. En este ejemplo, cada vez que un Manager necesita nueva información sobre el ordenador de un desarrollador, dos métodos tienen que crearse, el método de conveniencia en la clase Developer y el método delegado en la clase Computer.

Notas: debemos captar simultáneamente todas las dependencias que existen a través de múltiples clases.

En grandes aplicaciones, el número de clases implicadas y el recuento promedio de los métodos en las clases pueden ser mucho mas altos y vuelve ese tipo de valoraciones mas difícil al utilizar solo los métodos tradicionales.

Lograr buenos diseños en aplicaciones Web no es lo mismo que lograr buenos diseños en aplicaciones comunes, Hacer cambios en aplicaciones Web puede llevar a hacer cambios en varias partes, ya que una aplicación Web posee mas de una capa. Por ello a continuación nuestra idea es poder llevar el concepto de refactoring a las aplicaciones Web.

6.3. Refactoring en aplicaciones Web

Llevar una aplicación Web tradicional a una Aplicación Web Rica, si bien implica tener que "enriquecer" cada una de las funcionalidades se puede hacer en proceso de migración de a pequeños pasos, por varias razones, entre ellas podemos mencionar:

Reducción de los tiempos de desarrollo: migrar una aplicación en pequeñas etapas es menos costoso, no es necesario esperar tiempos largos para poner en producción una aplicación, lo ideal es hacer pequeños cambios para mejorar la funcionalidad que tiene actualmente.

Introducción de errores: si hacemos una migración de pequeños pasos hay menos probabilidades de introducir errores, o mejor dicho, es mas fácil identificarlos, porque lo que estamos haciendo son cambios puntuales y no grandes cambios donde podría ser costoso identificarlos

Recursos: Migrar una aplicación por completo donde involucra por lo general cambios de tecnología, arquitectura, etc., implica contar con una planta de recursos humanos que puedan que llevar adelante un proceso de estas magnitudes, en cambio, si los hacemos en una menor escala, dependiendo del tipo de aplicación no es necesario un equipo significativo.

Al momento de empezar la migración se puede hacer foco en modificar las funcionalidades que el usuario mas utiliza, este podría ser el índice de la aplicación, la forma de agregar productos a un carrito de compras, la forma de enviar un email, la forma de presentación, etc.

Hacer todos los cambios de una sola vez puede ser muy costoso y producir un efecto adverso o de rechazo para el usuario de la aplicación al intentar usarla, si los cambios son muy pronunciados, el usuario podría sentir rechazo a el uso de la aplicación, si en cambio introduzco un pequeño cambio, modifico una funcionalidad o bien utilizo el concepto de refactoring sobre una funcionalidad que ya existía, esto es modificar un control existente, donde el fin sea el mismo

pero mejorado, tanto en la utilización de recursos como para el uso cotidiano del usuario.

El impacto en cuanto a la usabilidad del usuario, podría verse afectada, la idea de un cambio repentino en una aplicación puede acarrear problemas serios de usabilidad, un usuario podría encontrarse “perdido” literalmente si los cambios son muy “groseros”.

6.3.1. Como podemos aplicarlo

El refactoring puede ser aplicado a nivel modelo o implementación.
El refactoring a nivel de implementación es un proceso similar a hacer un refactoring a la estructura del código, los refactorings pueden hacerse no solo en un modelo orientado a objetos, también se puede hacer refactoring de código de lenguajes Web tales como HTML, XML, Javascript, etc.
Un refactoring puede ser aplicado en alguna parte del modelo de diseño de la aplicación Web como por ejemplo a nivel capa de aplicación, modelo navegacional, o nivel de presentación, aquí ignoraremos el refactoring a nivel aplicación ya que es lo mismo que el refactoring tradicional. Y nos enfocaremos en los otros dos tipos de modelos.
Definiremos el “Web model refactoring” como el efecto que puede causar en el cual la aplicación presenta los contenidos, y provee las funcionalidades de interacción, ósea refactoring en los modelos de navegación y modelo de presentación.
Para entrar en una discusión concreta, para cada uno de los dos modelos, consideraremos las propiedades de la aplicación Web y definiremos sobre que aspectos podrimos aplicar el concepto de refactoring.

Para el modelo navegacional un refactoring involucraría cambiar las siguientes propiedades:

- Contenidos incluidos dentro de un nodo
- Links salientes dentro de un nodo
- Topología de navegación asociada a un conjunto de nodos(visita guiada, índice, todos con todos, un mix de todos los nombrados)
- Habilitar operaciones a los usuarios por nodo
- Ruta navegación asociado a la ejecución de uno o mas tareas de usuario o procesos de negocios

Para el modelo de presentación se pueden incluir los siguientes refactoring’s:

- Encuadre general de la pagina
- Look and feel de un elemento grafico presentado por la pagina
- Formatear el contenido (Hoja de estilo asociado a la pagina)
- Tipos, posición y números de sección in cada pagina

- Información, por ejemplo de los nodos, presentadas en las secciones de la pagina
- Reflejar Transformaciones de interfaces como resultado de la interacción de un usuario.

Cada tipo de “model refactoring” que listamos puede ser mapeado a uno o más refactoring a nivel presentación. Conviene señalar que mientras el refactoring convencional esta orientado a mejorar la calidad del código tanto como su mantenimiento, los Web refactoring impactan directamente en la usabilidad de la aplicación Web proporcionándole al usuario experiencia mientras navega la aplicación Web.

Para precisar la descripción del refactoring, es importante definir que comportamiento queremos transformar y el comportamiento que la aplicación Web deberá proporcionar.

7. Framework Conceptual para migrar aplicaciones convencionales a RIA

7.1. Introducción

En los capítulos anteriores fuimos viendo conceptos como refactoring, aplicaciones RIA, Web Patterns, las aplicaciones Web tradicionales y sus falencias y AJAX entre otros, nosotros intentaremos sacar provecho de las ventajas que ofrece cada uno de estos conceptos, para poder llevar nuestras aplicaciones Web tradicionales a aplicaciones RIA.

Haremos foco en los distintos aspectos que podemos mejorar utilizando lo antes mencionado y darle al usuario aplicaciones con nuevas características más atractivas, eficientes. Que hacen a una mejor usabilidad de la misma.

A partir de acá intentaremos desplegar algunos "concejos" que los desarrolladores deberían tener en cuenta al momento de empezar a llevar sus aplicaciones Web hacia una aplicación RIA.

Debemos tener muy en cuenta que es imposible empezar una migración sin la utilización de un Framework RIA para poder empezar a evitar las recargas de pagina y agilizar las funcionalidades de los componentes, cargarlos de información, cargarlos de funcionalidad, o mejorar la ya existente.

A continuación se detallan una serie de conceptos que se deberán tener en cuenta al momento de iniciar un proceso de cambio en sus aplicaciones Web.

7.2. Cambios en el Modelo Navegacional

El modelo navegacional de una aplicación Web es usualmente definida como un diagrama de clases navegacionales, el diagrama se asemeja a un diagrama de clases UML donde las cajas son clases de nodos, las asociaciones son links y los índices son un tipo particular de nodos que son definidos con el fin de simplificar la navegación, sin embargo, los refactorings sobre la estructura navegacional sigue siendo un diagrama de clases navegacionales con cambios en alguna de sus clases o cambios de sintaxis.
El comportamiento de una aplicación Web, como especificamos en este nivel, es dado por el conjunto de operaciones o acciones disponibles en cada nodo pero también será dado por el conjunto de links que permiten al usuario navegar a través de los nodos. Los refactorings a este nivel deberán preservar:

- El conjunto de operaciones posibles y la semántica de cada operación
- La "navegabilidad", la cual es definida como el conjunto de paginas disponibles.

Teniendo en cuenta esto podremos definir “refactorings navegacionales”.
Los refactoring navegacionales en un modelo Web son transformaciones aplicadas al diagrama de clases navegacional que preserva las operaciones posibles y la navegabilidad.
Algunos ejemplos de refactoring navegacionales son:

- Agregar información u operaciones a un nodo además de las que ya tenía en el modelo conceptual.
- Agregar links Nuevos entre paginas
- Remover un link que en la actualidad hace que me lleve a una pagina que no existe.
- Agregar una nueva página o remover paginas que ya están desactualizadas.

7.3. Cambios en la Capa de presentación

El modelo de presentación describe la “interfase abstracta” de la aplicación especificando como navegar los objetos y la funcionalidad de la aplicación que el usuario percibirá. Mas allá de eso, hay que hacer foco en varios tipos de funcionalidad que el usuario puede ejecutar a través de los elementos de una interfaz con respecto a la información intercambiada entre el usuario y la aplicación Web. El modelo de presentación es compuesto de páginas y componentes gráficos que describen el lock and feel de las páginas, sin embargo los refactoring a este nivel pueden ser:

- Cambiar el look and feel de una pagina intercambiando los componentes gráficos
- Agregar información o operaciones disponibles sobre el nodo
- Agregar o cambiar los efectos gráficos disponibles sobre la pagina

Nótese que la mayoría de los refactoring navegaciones implican algún tipo de cambio en la interfase mientras que los refactoring sobre las interfaces no implican cambios en la estructura navegacional.

7.4. Refactorizar utilizando patrones Web

Los patrones Web suelen se muy útiles al momento de hacer cambios en nuestra aplicación, nos proporcionan una serie de pasos o cambios que podemos realizar para lograr el resultado deseado sin desviarnos del mismo. Muchas veces aplicamos patrones sin darnos cuenta, el concepto de Web pattern emergió del concepto de patrones de hipermedia estudiado allá por los ’90 [9; 20].

Los Web patterns son similares a diseñar con patrones en aplicaciones convencionales, ellos son utilizados para solucionar un problema específico de una aplicación. Existe una lista bastante completa de Web patterns para poder solucionar determinados problemas. [3, 23, 24].

Cada tipo de problema tiene una solucion determinada y estos catalogos de patterns pueden ayudarnos a atacar estos problemas de una manera mucho más estrategica.

Teniendo en cuenta este concepto nuestra idea es usar el mismo concepto "Refactoring to Patterns" [8], nuestra intención es lograr realizar mejoras sobre un aplicación aplicando el concepto de refactoring, esto es mejorar una determinada funcionalidad sin cambiar el comportamiento de la misma. Para esto existen una serie de patrones Web que podrían aplicar a los casos donde nosotros queremos aplicar un cambio o mejora. A este "tipo" de cambios, el autor Karievsky también los llama "*pattern-directed refactorings*". Estos también discuten un problema que el patron Web ayudan a solucionar con sus pro y sus contras de aplicar el mismo.

7.5. Refactoring's Atómicos

Si bien todos sabemos que significa atómico, unido a la palabra refactoring, hace necesaria una explicación, un refactoring atómico hace hincapié en la cantidad de cambios que voy a realizar sobre la funcionalidad que queremos "retocar", atómico, quiere decir que el cambio apunta a cambiar una solo tipo ya sea para mejorar el aspecto, mejorar una funcionalidad, la navegación ,etc.

A continuación hemos definido una serie de cambios que creemos convenientes, útiles y que ayudarían al usuario cumplir las tares con la aplicación.

7.5.1. Agregar información

Motivación: estudios recientes sobre la usabilidad en las aplicaciones Web demuestran que muchas veces la información que se muestra en una página no es suficiente para la toma de decisiones del usuario. Por ello un cambio importante seria agregar información ya sea proveniente del modelo de aplicación o de cualquier otro recurso.
también podría ser información agregada para hacer un seguimiento de clientes o vendedores, también puede ser información de ayuda obtenida del mismo modelo de navegación.

Este refactoring puede ser utilizado para introducir patrones como "*Clean Product Details* [3]" para agregar detalles a los productos en una aplicación de e-commerse.
Con el propósito de atraer cliente, podemos introducir recomendación de productos personalizada o información de ranking de los mismos.
Para mejorar la navegación, podemos introducir el pattern (*Active Reference* [9]), por ejemplo para mostrar la ruta completa del camino recorrido para llegar al producto.

Mecanismo: el mecanismo puede variar de acuerdo a la elección que hagamos de la opciones antes mencionadas, en el caso mas general, agregamos un atributo con la información que deseemos agregar, a la clase del nodo en el modelo de navegación.
Si la información extraída de el modelo de aplicación, la atachamos al atributo, describiendo el mapeo a el modelo de aplicación [19]
Si usamos *Active Reference* [9], agregamos un índice a la clase del nodo tal que la pagina actual es remarcada en el índice

Ejemplo: este refactoring es aplicado para transformar la pagina que aparece en la figura 1 en un índice que aparece en la pagina 2, en este caso la información es agregada a cada entrada de el índice; la información agregada incluye la portada de CD, el precio, rating, información de venta, links a la lista vendedores, año de edición, etc.

Impacto*:* este es un refactoring que involucra agregar atributos a la clase del nodo en el modelo de navegación, sin embargo, también produce cambios en la en la capa de presentación, ya que deberemos agregar un widget para mostrar la nueva información.

7.5.2. Agregar Operación

Motivación: Las operaciones siempre deberían aparecer cerca de los datos en los cuales funcionan, y las aplicaciones de Web deberían ser diseñadas con eso en mente. Sin embargo, las operaciones pueden ser agregadas posteriormente a las páginas Web por varias razones, entre ellas podemos destacar:

- Como el resultado de una operación acrecentó el modelo aplicativo por un requisito nuevo
- Para acelerar un proceso check-out de una aplicación de e-commerse
- Para proveer *Printable Pages* [3]

Las operaciones también pueden ser agregadas a cada entrada de un índice con el fin de que el usuario no necesite navegar hacia el nodo, pudiendo hacerlo directamente sobre la entrada

Mecanismo: agregar una operación a una determinada clase en el modelo de navegación. Podemos notar que la operación deberá ser disponible en el diseño del modelo de la aplicación.

Impacto*:* agregar operaciones a el modelo navegacional requiere que modifiquemos la interfase agregando un widget para poder incluirla, nótese que este refactoring si bien comienza realizando cambios sobre la capa de navegación, termina con la modificación de la capa de presentación (agregar un botón o agregar una operación en la interfase), por ultimo debemos mencionar que agregar operaciones a las entradas nos posibilita interfaces de usuario mas accesibles e interactivas.

Ejemplo: en el e-shop de libros de Amazon, el proceso de compra ha sido mejorado considerablemente logrando mayor velocidad. Cuando un ítem es agregado a la cesta de compra, la cesta tiene un botón más que dice “comprar ahora con un solo click”. Al hacer click sobre ese botón le permite al usuario cargar la información de pago una sola vez y no cada vez que compra algo.

7.5.3. Anticipar Target

Motivación*:* Diferentes testeos en el uso de la aplicación pueden mostrar que los usuarios van y vienen repetidamente después de entrar en un link. La razón de esto es generalmente es porque el “target” del link no era el que el usuario esperaba. Estas idas y vueltas terminan por lo general en una rápida frustración y abandono del sitio.

Para prevenir esto, el target del link deberá indicarnos de alguna manera, mejor dicho explicarnos mejor a que se refiere el mismo. Una solución es darle al link del tag un nombre mas representativo, cuando un sitio Web esta dirigido a una comunidad cerrada con un vocabulario común, este nombramiento no causara demasiados problemas, ahora el problema se presenta cuando los usuarios de una determinada aplicación pertenecen a distintas generaciones o bien distintas culturas, o bien diferentes comunidades, para este caso lo mejor seria que el target anticipe cual es el destino del mismo para no causar un efecto de ida y vuelta no deseado.

Mecanismo*:* hay diferentes formas de lograr que un link me de una breve descripción de hacia donde me dirigirá*.* A continuación describiremos algunas soluciones que involucran patrones de hipermedia

Solución 1: “*Mostrar el target”***:** Agregar un script al “anchor” del link que cuando el mouse sea pasado por encima muestra una pequeña descripción del

target. Este es una refactoring sobre la interfase y usualmente se implementa usando AJAX. Esta solución se recomienda cuando el link lleva a una página fuera de la aplicación

Solución 2: "*Agregar información a los links"*. Esta solución es una variación del refactoring "agregar información"; en este caso la información es agregada en un nodo distinto y es mostrado al posicionarse sobre el link, la información mostrada en el target deberá ser elegida cuidadosamente para no provocar una activación falsa del link.
Este refactoring aplica al modelo de navegación, pero obviamente trae consecuencias sobre la capa de presentación para reflejar estos cambios.
En este punto podemos usar diferentes alternativas, desde agregar widgets al posicionarme sobre un link a introducir patterns tales como "información por demanda [9]".

Ejemplo:

Podemos ver como al pasar el Mouse por encima del nombre de la película se despliega un widget con una imagen y una reseña de la película, además muestra información del director y el género.

Figure 3. Muestra un ejemplo de Dr. Dobb's Portal (http://www.ddj.com) que usa el refactoring "agregar información al target" para los diferentes links en el mismo área, usando el pattern "information on demand" [9].

Impacto*:* nótese que este refactoring tiene dos mecánicas alternativas, una al nivel presentación, y la otra a nivel navegación. Si aplicamos la segunda opción veremos que nos llevara a realizar cambios a nivel presentación para reflejar las transformaciones navegacionales.

7.6. Refactorings compuestos

Cada uno de los refactoring antes presentados son autocontenidos y pueden ayudar a mejorar la usabilidad de la aplicación. Sin embargo una sabia composición de refactorings atómicos pueden implicar una transformación no trivial en un alto impacto en la usabilidad, describiendo paso por paso la secuencia.

Para ilustrar la composición de refactorings a continuación se explica como enriquecer un índice en tres pasos usando los refactoring atómicos que explicamos anteriormente.

7.6.1. Enriquecimiento de índices

Motivación*:* en muchas aplicaciones Web que navegamos no solamente llegamos a la información a través de objetos sino también operando con ellos. Si bien después de aplicar el pattern, los links nos dan una pista sobre el destino de los mismos para decidir si los navegamos o no. Muchas veces estas pistas no son suficientes para tomas decisiones de navegación por ejemplo cuando tengo una lista de links como índice.
Una vez que decidimos que link seleccionar, debemos buscar reducir el numero de navegaciones sin encontrar el resultado que buscábamos.
Una solución para enriquecer un índice es agregarle información y operaciones a cada uno de los ítems, sin embargo los espacios entre ítems del menú generalmente son estrechos, al momento de armar la capa de presentación debemos tener en cuenta esta característica importante.

Let It Bleed [DSD] ~ The Rolling Stones
Exile on Main St. ~ The Rolling Stones
Beggars Banquet ~ The Rolling Stones
Some Girls ~ The Rolling Stones
Goats Head Soup ~ The Rolling Stones
Tattoo You ~ The Rolling Stones
Get Yer Ya-Ya's Out! ~ The Rolling Stones
Flowers ~ The Rolling Stones

Figura 1: índice simple

Mecanismo*:* supongamos que empezamos con un índice simple como el de la figura 1 que provee acceso a los productos. Los siguientes pasos muestran como usar refactoring atómicos para mejorarlo:

1) Aplicar “agregar información” para anticipar el destino del de cada elemento del índice. La información es obtenida del nodo destino correspondiente a la entrada del índice. La transformación concreta en términos del diagrama navegacional de OOHDM es cambiar el anchor que me permite navegar al target especificado a un estructura mas compleja y además un query para

obtener la información hacia donde me lleva el link. El resultado final después de aplicar este paso se muestra en la Figura 2

Figura 2: este es el resultado de agregar información a los ítems, imágenes, precio, ratings, etc.

2) tenemos que aplicar el pattern “aplicar operación” a cada elemento del Index. Estas operaciones agregadas serán disponibles inmediatamente al usuario al posicionarse sobre un ítem.
El resultado final del refactoring puede verse en la figura 4.

3) después de aplicar el paso 1 y paso 2, el índice seguramente ocupara mas espacio en la pagina, por lo que nuestro refactoring como consecuencia nos llevara a modificar la pagina donde se muestra el índice, para esto se proponen distintas alternativas para concluir el refactoring

Mostrar información por demanda
Esta solución esta dada por la aplicación del pattern *Information on Demand* [9], usando este pattern, la misma sección de la pagina es usada para mostrar información de las diferentes entradas, el resultado de esto podemos verlos en la figura 3

Aplicar pattern "anticipación del target"
Aplicando este pattern se agrega un control a cada entrada del índice o un widget para mostrar la anticipación de a donde no esta llevando la entrada, cuando el Mouse sea posicionado sobre la entrada aparecerá un popup con información y operaciones agregadas en los pasos 1 y 2. El inconveniente de esto es que los popup's pueden estar bloqueados o causar malestar en algunos usuarios, podemos ver el ejemplo en la figura 5

Scrolling
Usar scrolling vertical y horizontal permite a el índice coexistir con otros widgets de la pagina (figura 5)

Dividir la vista
Divide las entradas de un índice en varias páginas, permitiendo al usuario navegarlos secuencialmente, un ejemplo de esto puede ser el resultado de las búsquedas de Google.

Ejemplo: las aplicaciones de E-commerce usualmente proveen recomendaciones de sus productos como una forma efectiva de ofertar.
Esta es una característica que los usuarios normalmente desearían de las aplicaciones, En un sitio Web emergente, las recomendaciones pueden ser simplemente una lista de los productos con un título y un enlace para la página del producto, un ejemplo de esto es el índice de la figura 1.

Podemos ver los resultados intermedios de la aplicación de cada uno de los pasos de este refactoring en la diferentes versiones de "recomendaciones" de Amazon, comenzando por la figura 1, si aplicamos el pattern "agregar información" donde unos de los datos agregados es el precio, rating, y una imagen de la tapa del disco, llevamos a un índice mucho mas rico como el de la figura 2.
Una vez que agregamos información, podemos usar el pattern "agregar operaciones" y "agregar al carrito" y vemos que nos queda algo como la figura 3.

Aplicando todo lo antes mencionado el índice se convierte en una lista muy larga, entonces finalmente, el espacio debe ser reducido, entonces debemos aplicar refactoring sobre la presentación, aquí podríamos usar la instrucción de información por demanda y "scrolling" como se muestra en el paso tres, el resultado final lo podemos ver en la figura 5.

Figure 4. Esta imagen muestra el resultado de agregar operación a los ítems de la figura 1

Figure 5. Índice enriquecido

Impacto*:* el enriquecimiento de índices esta compuesto de refactorings que aplican a nivel de navegación y a nivel de presentación. Además, esto muestra que podemos componer refactorings en diferentes formas: una de ellas puede ser conjunción, donde todos los refactorings son aplicados en un orden particular, o de modo disyuntivo, donde aplicamos los refactorings alternativamente.

7.6.2. Refinación Búsqueda

Motivación: Muchas veces las búsquedas por palabras claves retornan demasiados resultados, por supuesto también demasiadas páginas con resultados, muchos de los cuales no nos son útiles. Muchas veces los resultados que nosotros buscamos queremos que pertenezcan a una determinada región, estado, o que se encuentre en un determinado rango de precios, que pertenezcan a una determinada marca, un determinado modelo, etc.
Si organizamos y/o clasificamos los productos podemos reducir los tiempos y/o recursos para lograr encontrar lo que buscamos sin perdernos en una lista amplia de resultados, dado que los filtros son muy genéricos. La mayoría de las aplicaciones proporciona búsquedas básicas filtrando por strings como muestra la figura 1, una búsqueda así permitiría ingresar filtros que entre si no tuviesen ningún tipo de relación.

Mecanismo: hacer un refactoring de este estilo implica usar un refactoring atómico como puede ser “agregar operación” , aunque esta operación implique la combinación del/los nodo/s en cuestión con el componente que provea la funcionalidad de filtrado de resultados, mas allá de hacer cambios en la capa de presentación donde agregaremos un panel con la lógica, necesitaremos revisar la lógica en la forma de almacenar y de recuperar los resultados , y por supuesto agregar a la capa de navegación el componente con la lógica para refinar los resultados es los ideal para estos casos es agrupar cada uno de los objetos, en categorías.
Proveer una funcionalidad con estas características a un sitio implica rediseñar la capa de presentación con un panel donde se muestren los diferentes filtros que podremos aplicar post-búsqueda de productos como se muestran en la figura 3, este panel mostrara los diferentes grupos de todos los objetos resultados de la búsqueda, esto también facilita poder seleccionar uno o mas grupos y volver a realizar la búsqueda con los filtros nuevos, acotando la búsqueda a un resultado mas pequeño.
Este mecanismo me da la posibilidad de hacer la búsqueda de dos maneras, una mucho más simple que seria navegar los distintos grupos

Nota: Cada uno de los filtros dentro del panel pueden estar acompañados de la cantidad de objetos que corresponden al mismo dentro de la búsqueda actual, esto facilita mas aun la refinación de la búsqueda.

Impacto: usando “Refinación de búsqueda” permitiremos al usuario encontrar lo que busca de una manera mucho más fácil y sencilla, sin necesidad de frustrarse y causar una mala experiencia sobre la aplicación. Antes para buscar lo que necesitábamos debíamos usar la famosa búsqueda avanzada, que si bien no hay porque no tenerla en cuenta, no nos permitía la funcionalidad de un panel de refinación de búsqueda.

Ejemplo:

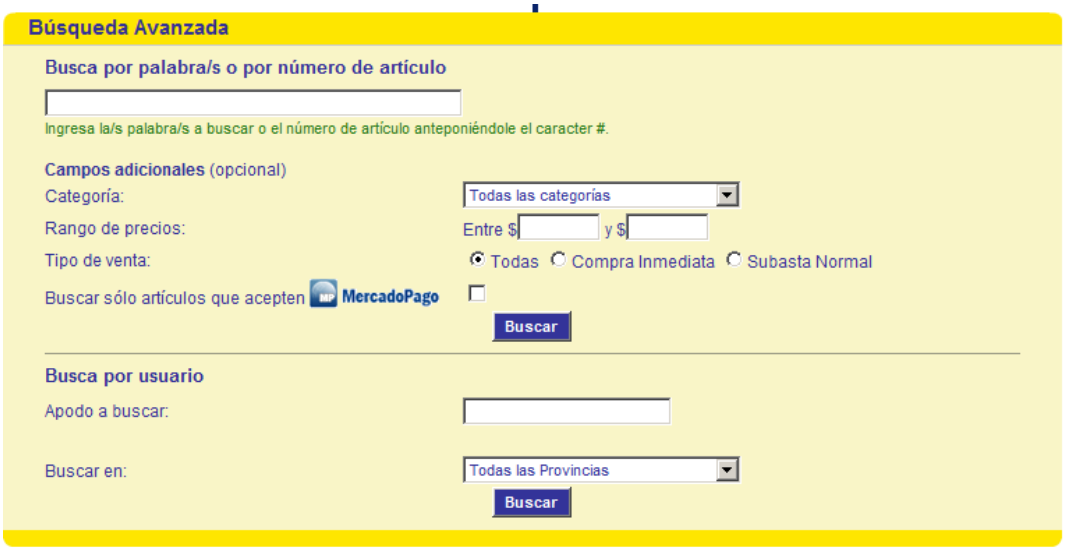


Figura 1: búsqueda básica

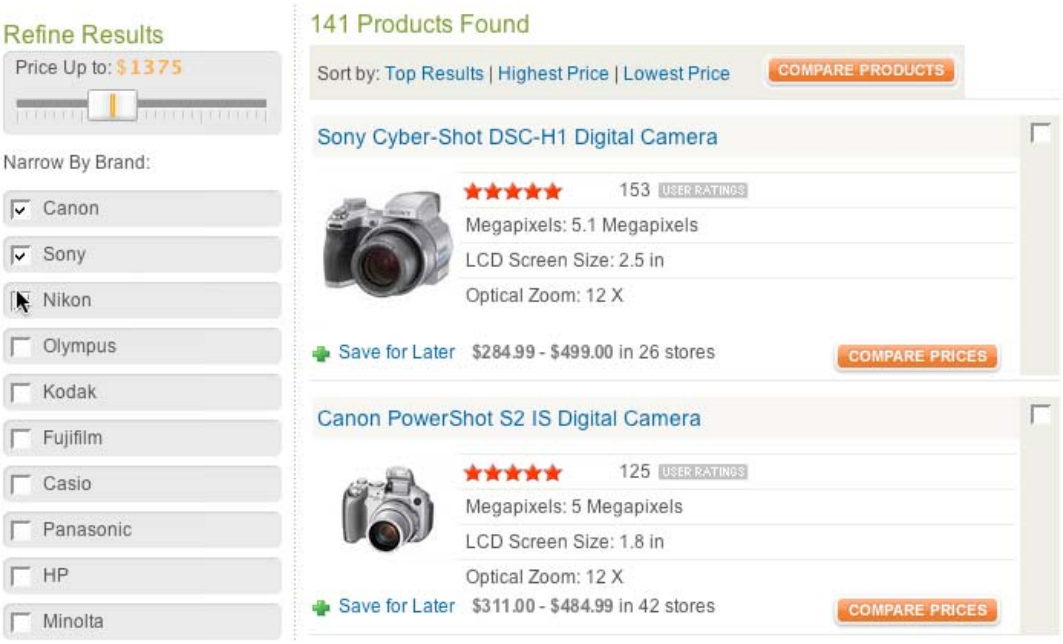


Figura 2: Búsqueda interactiva

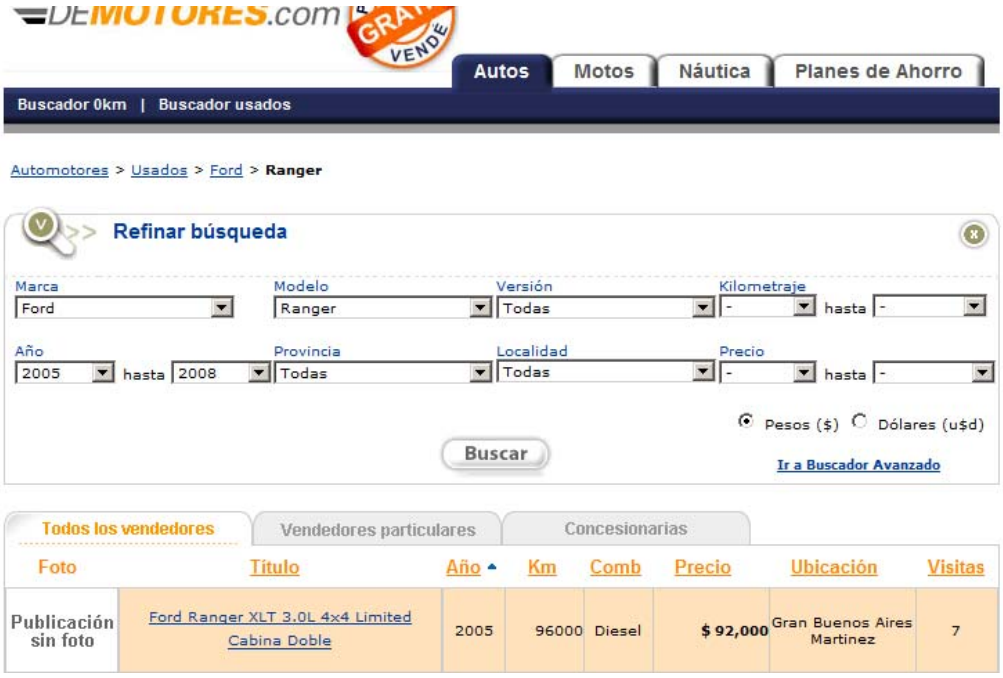


Figura 3: permite ingresar filtros validos de búsqueda

7.6.3. Live Suggest (de funcionalidad)

Motivación: supongamos que deseamos enviar un mail o hacer una búsqueda por alguna palabra clave del sitio. Basémonos en el primer caso, Para ingresar el remitente por lo general, necesitamos memorizarlo, o bien seleccionarlo de un popup con la lista de contactos (figura 1). Esta tarea podría ser mucho mas sencilla si ingresáramos la inicial del nombre o la primera letra de email y nos desplegara una lista con los posibles remitentes que se correspondan con las iniciales ingresadas., o bien cuando queramos realizar una búsqueda por alguna palabra clave, seria bueno que el input nos sugiera cual podría ser una palabra por la cual la búsqueda nos devuelva un resultado coherente.

Mecanismo: un refactoring de este estilo requeriría diseñar un componente que implemente un "*Reflector*" que cuando se produzca el textchanged haga una búsqueda y muestre los resultados desplegando un panel con los resultados a partir de el corner izquierdo inferior del textbox, al empezar a ingresar los caracteres de la palabra se cargarían los posibles resultados que concuerden con las primeras letras del nombre de un contacto o de una palabra calve o bien de una búsqueda anterior o de una dirección de email; Un componente de este estilo nos serviría para utilizarlo en diferentes lugares del sitio con distintas funcionalidades (sugerir un email, sugerir una búsqueda por una palabra clave, etc.).
Obviamente deberemos modificar la capa de aplicación para que implementar el refactor y la búsqueda de las posibles sugerencias a través de este.

Impacto: un componente de este estilo nos facilitaría muchas tareas y nos haría reducir los tiempos de las que estamos realizando.
El ejemplo de la imagen que se encuentra a continuación podemos ver que a medida que vamos escribiendo las primeras letras de cómo creemos que empieza la cuenta de email del destinatario nos despliega un panel con las posibles cuentas que se corresponden con la letras ingresadas. La búsqueda la realiza sobre la lista de contactos o bien sobre el destinatario o un emisor que figure en la bandeja de entrada o salida.
Si bien su uso es muy sencillo, la lógica que oculta detrás no es tan simple. Pero se podría generalizar lo más posibles para su posible reuso.
En el segundo ejemplo mostramos como seria una sugerencia de búsqueda que me retorne resultados. Vemos como al ingresar un

Ejemplo:

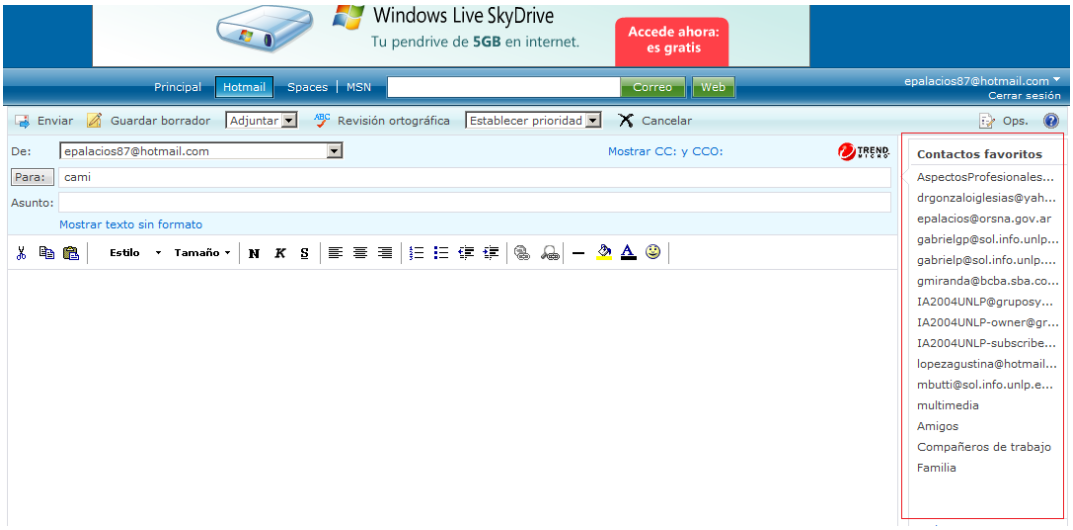


Figura 1: como podemos ver para seleccionar el receptor lo deberemos buscar y seleccionar del panel derecho remarcado con rojo

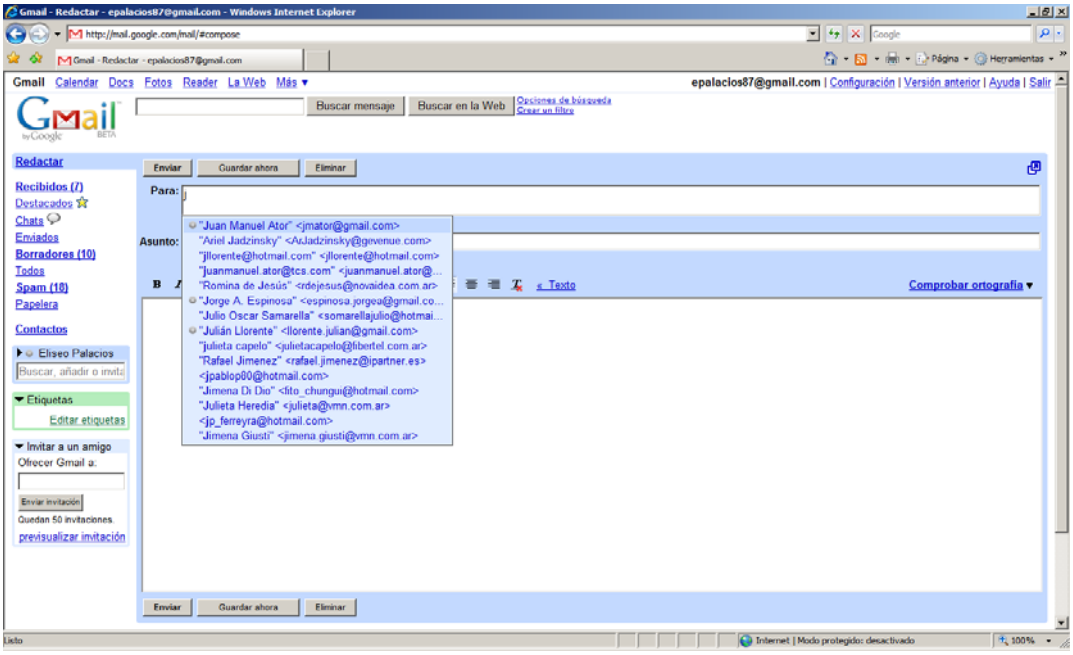


Figura 3: En este ejemplo podemos ver que al ingresar una inicial se despliegan un conjunto de posibles receptores del email, esta búsqueda la realiza sobre la lista de contactos del propietario de la cuenta email.

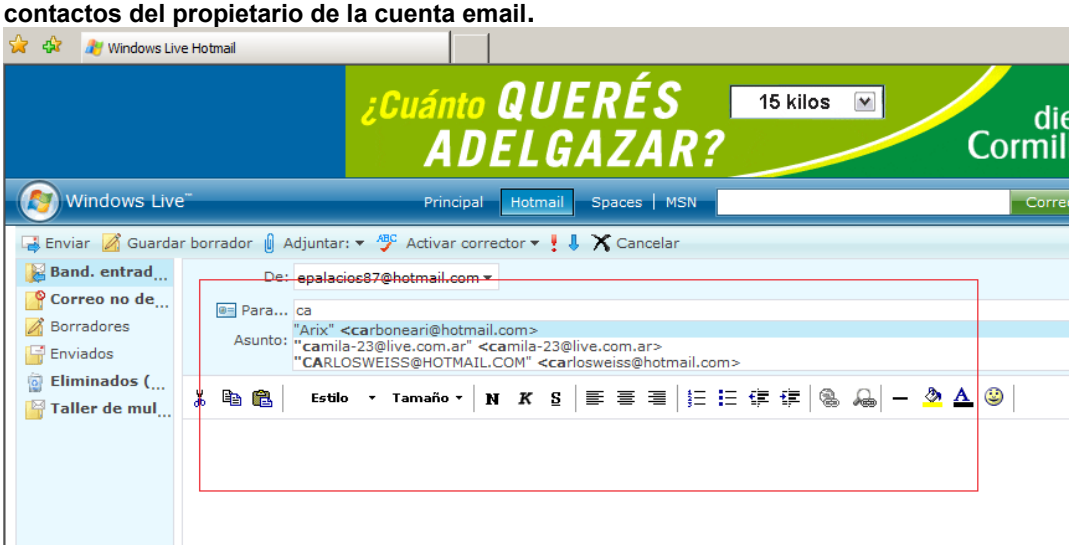


Figura 3: al ingresar una o dos letras automáticamente se despliega un widget con sugerencias de los posibles receptores del mail.

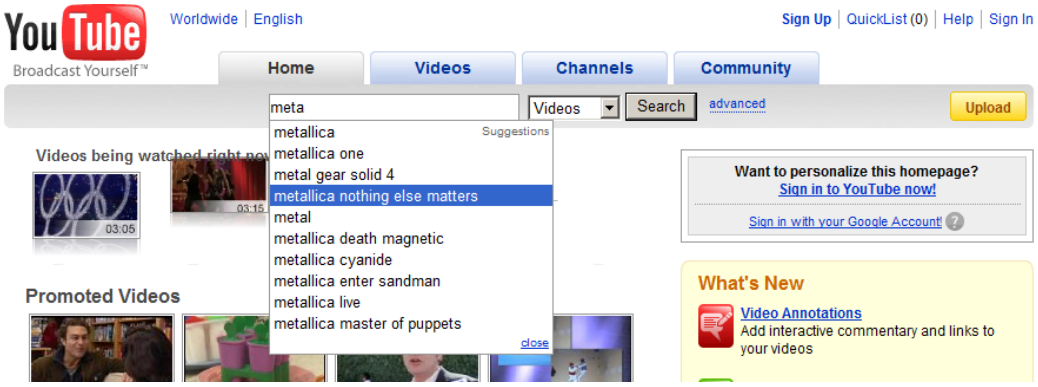


Figura 4: En este ejemplo podemos ver que al ingresar la palabra meta automáticamente me señala que búsquedas con resultados podría realizar

7.6.4. AutoGuardado

Motivación: muchas veces perdemos información por diferentes factores externos, ya sea cuando estamos escribiendo un email o algún tipo de documento.
Muchos editores de texto tienen la inteligencia para producir un autoguardado cada determinada cantidad de tiempo o ante un cierre inesperado de la aplicación. Teniendo en cuenta esto y las posibilidades que nos proveen hoy las herramientas de desarrollo, sin grandes cambios podríamos desarrollar una funcionalidad con estas características haciendo más confiables las aplicaciones, en la actualidad sacando al gestor de email *Gmail* casi ninguna aplicación provee esta funcionalidad

Mecanismo: debemos agregar funcionalidad para que después de cierto tiempo produzca un autoguardado.
Podemos partir de una opción de configuración donde le digamos a nuestro editor ya sea de notas o bien de mails que cuando se cumpla cierto periodo de tiempo guarde nuestro contenido dentro de una carpeta borradores predeterminada.
A continuación se detallan las distintas consideraciones que deberemos tener en cuenta para llevar a cabo el refactoring

Guardado por tiempo
Si después de cierto tiempo no fue enviado el mensaje el mismo se guardara en la carpeta borradores

Sobrescribir si ya existía
Si el borrador para ese email ya existía se sobrescribe con el ultimo contenido

Eliminar si fue enviado
Si este luego es enviado, el borrador se autoeliminara

Impacto: este es un refactoring que muchas veces puede pasar desapercibido pero suele ser de gran funcionalidad ya que si por esas causalidades apretamos un botón que no correspondía o se colgó la maquina o se corto la luz. Cuando iniciemos sesión nuevamente podremos acceder al contenido escrito anteriormente dentro de una carpeta que podría ser borradores, hoy en día con Ajax esta funcionalidad pasa a ser una simple función javascript que pasado cierto periodo de tiempo de se ejecute una función de “autoguardado”. Gmail provee una funcionalidad con estas características y después de cierto tiempo sin interacción produce un autoguardado.
A continuación la imagen muestra un ejemplo

Ejemplo:

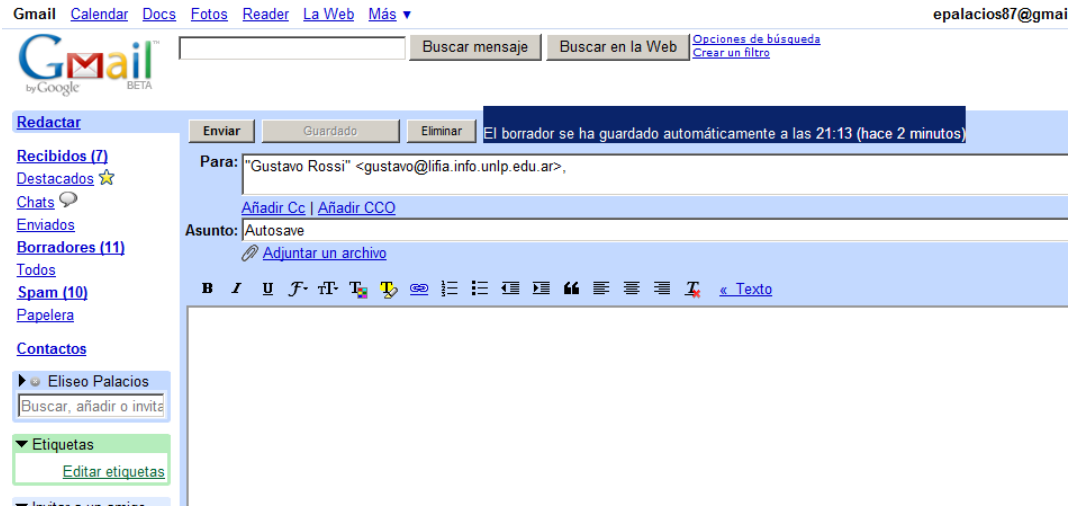


Figura 1: Después de unos instantes sin interacción se produce el autoguardado

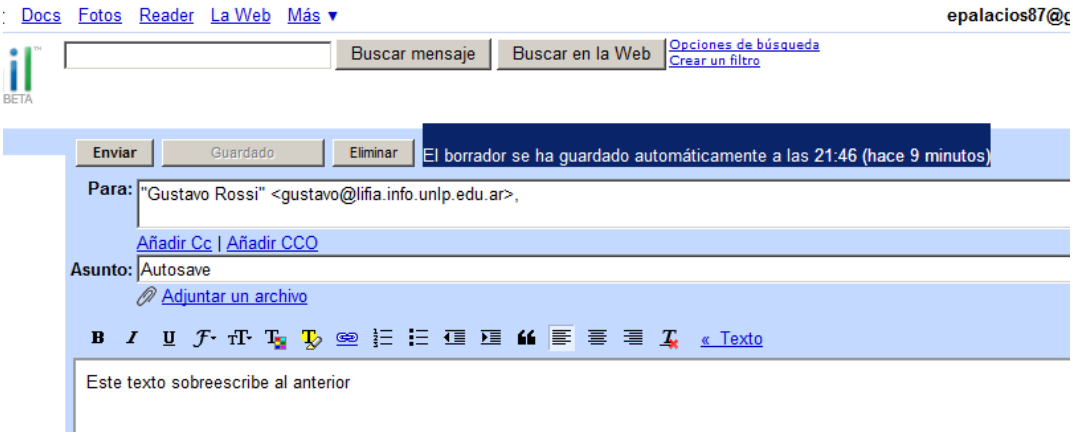


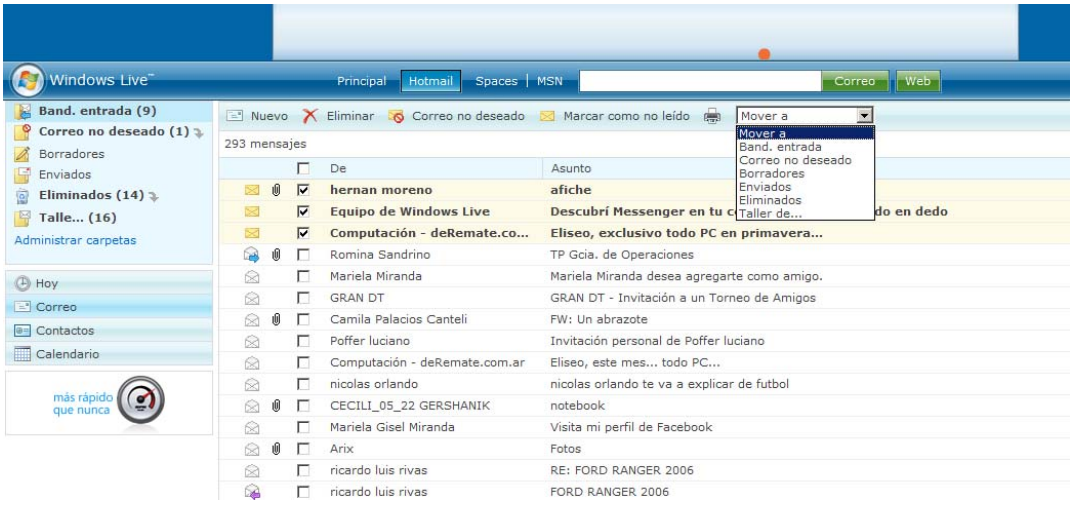
Figura 2: Este sobrescribe el borrador anterior

7.6.5. Drag and Drop (agregar comportamiento)

Motivación: Muchas veces deseamos tener organizados nuestros contenidos, mails, carpetas, documentos, o bien cuando estoy en alguna aplicación que me permita realizar una compra, seria muy interesante poder darle al usuario una aplicación que cumpla la funcionalidad que el usuario este esta acostumbrado/a a realizar, por ejemplo un supermercado, el cliente selecciona cosas y las deposita en un "carrito",en una aplicación Web podríamos brindar la misma funcionalidad si al usuario le proveemos una funcionalidad donde el usuario selecciona lo que va a comprar y lo arrastra hasta el "*carrito de comprar*".

Las aplicaciones de escritorio nos proveen distintos componentes para darle a una aplicación una funcionalidad de estas características, entre ellas podemos destacar un tree-view donde podemos mover, reacomodar los componentes de una rama a otra. Hoy en día con la aparición de Ajax, o mejor dicho con la combinación de tecnologías podemos proveerle a nuestras aplicaciones Web un aspecto mucho mas parecido a las aplicaciones de escritorio, y sobre todo la funcionalidad que hasta hace un tiempo era casi imposible implementarla en nuestras aplicaciones Web.

Veamos un ejemplo de cómo Hotmail llevo una funcionalidad que no tenia Drag & Drop a una que nos provee una funcionalidad que nos permite arrastrar los mails por ejemplo.



En este ejemplo podemos ver claramente como se hace para mover los mails entre las carpetas del sitio, se seleccionar los ítems que queremos mover y luego ejecutamos la acción desde un desplegable.

Otro ejemplo podría ser la forma en la que agregamos elementos al carrito de compras

La mayoría de la aplicaciones de e-commerce añade elementos al carrito haciendo click sobre una imagen al pie, o en algún lugar cerca de la información de producto. Seria ideal que pudiéramos arrastrar los elementos directamente hacia el carrito y poder sacarlo como si estuviéramos en un supermercado.
En la actualidad son pocas las aplicaciones que proveen funcionalidades con estas características.

Mecanismo: Involucrar una funcionalidad con estas características, requeriría que tengamos un modelo de objetos bien diseñado, ya que bien cuando arrastramos un ítem de lista, implícitamente estamos arrastrando un objeto, de una clase.
Para poder mover elementos gráficos, debemos darle comportamiento a la capa de presentación del objeto de navegación del modelo, programar la interfaz para poder almacenar, tanto del lugar donde van a ser alojados, como el lugar donde están alojados actualmente, ya que debemos sacarlos de un lugar y ponerlos en otro.
A la vista esta que no solo deberemos refactorizar la capa de aplicación sino la del modelos de objetos también.

Consideraciones:
Hay muchas veces que deberemos tener en cuenta que hay regiones de la pagina sobre las cuales no podemos hacer drag, también deberemos tener en cuenta la dirección del mousemove. En el ejemplo de la figura el evento mousemove tiene un restricción de que solo se puede mover en dirección izquierda. Si yo quisiese hacer drag y mover el Mouse hacia la derecha. El manejador no me lo permitiría; también esta de más decir que el contenedor posee elementos que deberán permitir operaciones de drag

Impacto:
Hacer drag and drop sobre elementos de una interfaz Web requería una lógica muy compleja.
Hoy en día con el auge de AJAX la implementación de una funcionalidad de este estilo suele ser más confiable en cuanto a resultados y fácil e implementar, su utilización no trae tantos bugs ni frustraciones al usarlo.
Para desarrollar una funcionalidad de esta magnitud hay una serie de cuestiones que deberemos tener en cuenta:

los eventos, a continuación se detalla la serie de eventos que una página deberá tener en cuanta para poder llevar a cabo un requerimiento de drag & drop sobre la misma:

- Generación de la pagina (dibuja los componentes)
- Posicionar el mouse sobre objeto sobre le cual se puede hacer drag
- Drag iniciado (mouse presionado y mover le mouse >= 3 pixeles)
- Drag sobre un target valido
- Drag sobre un invalido target área
- Drag sobre la posición original
- Drop aceptado
- Drop rechazado
- Drop vuelve a la posición original

Actores: Otro aspecto importante a tener en cuenta son los actores que participan en la funcionalidad de Drag & Drop de la pagina.

- Cursor
- Modulo o representación en miniatura del modulo
- Ubicación original del modulo sobre el cual se hace DRAG
- Posición sobre la cual hicimos drop

En la actualidad hay varia librerías para implementar la funcionalidad pero la mayoría de ellas con templan un mismo patrón que tienen en cuenta los mismos criterios al momento de implementar el diseño de una funcionalidad de Drag and drop.

Entre ellos podemos destacar los manejadores de eventos que necesitamos para determinar cuando comienza, cual es el estado y cuando termina el evento Drag and Drop

Evento mousedown: guarda las coordenadas donde comenzó el evento drag, setea la propiedad zindex de el recuadro, además cambia el estilo del elemento para indicar que un evento drag esta en proceso e inicializar algunas propiedades que se necesitaría para completar el evento drag and drop

Evento *mousemove*: este evento inspecciona continuamente las coordenadas del mouse teniendo en cuanta la propiedad *left* y la propiedad *Top.* Debemos tener en cuanta que estas propiedades no se llaman de la misma manera en los distintos navegadores por lo que necesitaríamos una lógica adicional para contemplar esto.

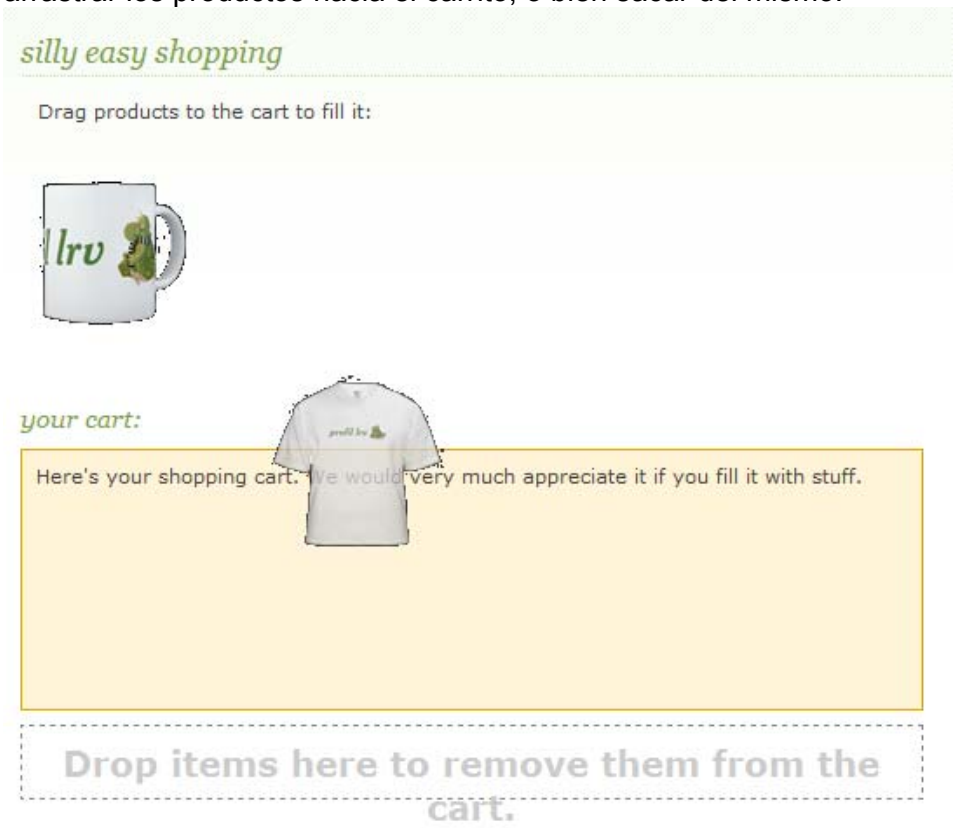
Evento *mouseup*: vuelve a cero las propiedades de estilo de l objeto y hace una limpieza de propiedades para poder volver a empezar con otra operación. Si se produce el mouseup sobre una región que no permitía un drag, todo queda como hasta el momento que se produjo el mousedown

Permite organizar los contenidos de una manera mucho mas fácil, ya que con un simple arrastre de objetos puedo tener mis contenidos en la rama adecuada de la aplicación, si necesidad de andar teniendo que seleccionar el origen y el destino y luego aplicar una acción a través de un menú desplegable. Esta característica acerca mucho mas una aplicación Web a una aplicación de escritorio, brindando al usuario un entorno mucho mas amigable y parecido a lo que esta acostumbrado a ver.

Ejemplo: en la imagen siguiente podemos ver como soluciono Hotmail el problema de mover los mails: mantuvo el estilo de carpetas a la izquierda pero agrego comportamiento ya que ahora podemos arrastrar nuestros mails de carpeta en carpeta

Figura 1: En este ejemplo podemos ver como podemos hacer drag and drop de mails

Ejemplo 2: en cuanto al problema del carrito de compras, a continuación se muestra un ejemplo practico para las aplicaciones de e-commerce, para poder arrastrar los productos hacia el carrito, o bien sacar del mismo.



Este ejemplo podemos ver como podría ser una compra online, seleccionamos lo que queremos comprar y lo arrastramos al carrito

Este seria el resultado de agregar productos al carrito

7.6.6. Agregar feedback

Motivación: Cuando estamos usando una aplicación con cada acción realizada la mayoría de la veces, ingresamos algún dato, enviamos un mail, modificamos la propiedad de un objeto o queremos que algo se vea de otra forma, en fin una infinidad de operaciones donde el usuario genera un evento sobre la aplicación; en muchos de estos eventos el usuario espera una respuesta por parte de la aplicación para poder tomar decisiones de si puede seguir, de si los datos que introdujo fueron validos, de si le gusta el color de auto que selecciono. Esta característica hace que la aplicación sea mucho más confiable, más amigable e interactiva. Si bien muchas aplicaciones intentan dar una funcionalidad con estas características, muchas lo hacen fuera de tiempo y forma, confundiendo al usuario.

Mecanismo: hay muchos eventos donde el usuario espera una respuesta por parte de la aplicación, entre ellos podríamos destacar:

- Cuando enviamos un email, el evento realiza una tarea pero por lo general el usuario no sabe si el servidor lo envío o no, en estos caso lo ideal es informarle al usuario de estado de la acción realizada para saber si el mail fue enviado por el servidor o no, para tener tranquilidad de que el mensaje va en camino

- Mostrar en línea si una contraseña “No” cumple con el patrón, la aplicación debería podernos informar en línea y no dejar que valide contra una fuente de datos directamente, por ejemplo si es requerido que los primeros cuatro dígitos sean numéricos. Porque dejar que valide si ya sabemos de antemano que el resultado será negativo.

- Cuando vamos a comprar algo muchas veces queremos ver cual es el color que nos gusta más de algo, por ejemplo un auto, un pantalón, una camisa, etc. En la actualidad se utiliza mucho esta técnica para mostrarle al usuario el resultado en línea de esto, esto también es una respuesta en línea hacia el usuario de un cambio que solicito

- En la actualidad hay muchas aplicaciones que nos proporcionan cierta información de cómo llegar a algún lado, mapas, estos mapas contienen mucha información y para mostrarlos en un pagina se les proporciona mucho “zoom in”; Pero el usuario necesita ver mas claramente el mapa por lo cual se le proporciona una herramienta para poder hacer “zoom out” y poder ver en detalle el mapa. El resultado de este zoom debería ser lo más rápido posible para que el usuario pueda ver en línea el recorrido que deberá realizar.

Ejemplos

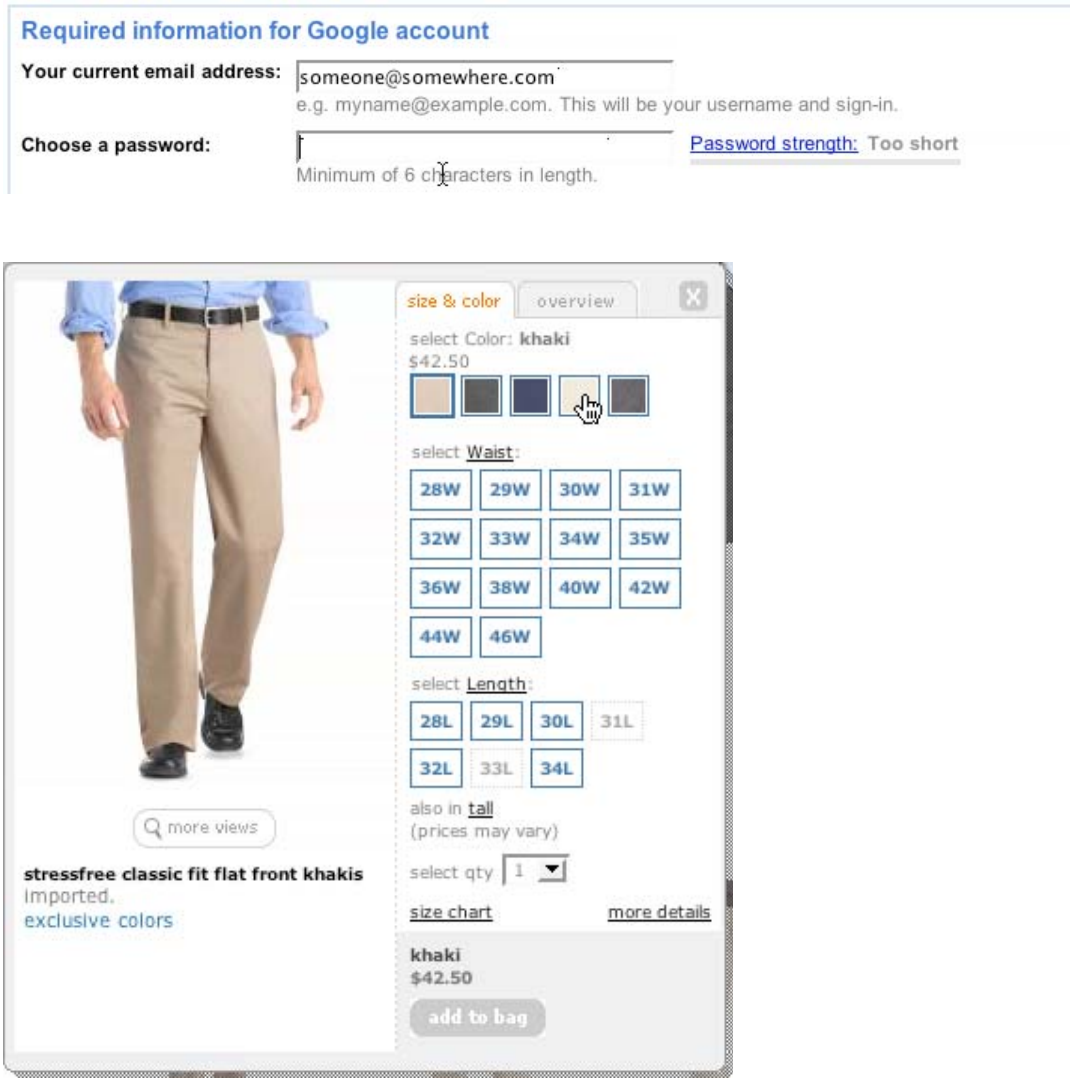


Figura 1: al seleccionar un color automáticamente vemos el cambio sobre la pantalla

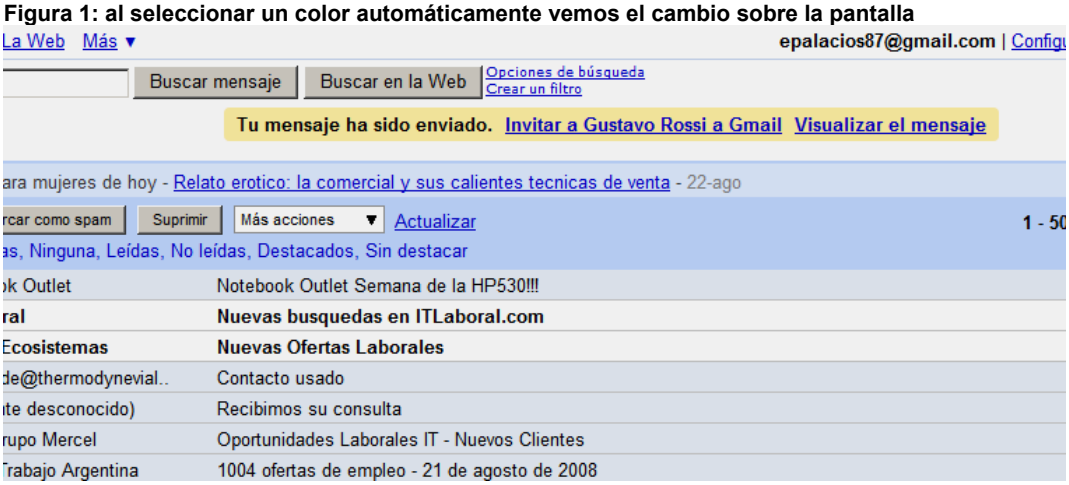


Figura 2: al enviar un email podemos ver que el sistema nos comunica cual fue el resultado de la acción realizada

7.6.7. PopUp's

Motivación: muchas veces es necesario realizar un cambio sobre alguna de las propiedades de un objeto, las cuales no aparecen sobre la página en cuestión, para estos casos lo mejor siempre es levantar un popup que permita realizar los cambios.
El problema de esto radica en que los popups son manejados de manera distinta a través de los distintos navegadores, lo que hace difícil implementar una lógica con estas características.
Otro problema importante que surge es como lograr que el usuario abra un popup y no se lo olvide abierto, con datos cargados sin distraerse con el flujo de la página principal.

Las siguientes situaciones solucionan muestran un marco de los ejemplos donde la mejor solución son los popups:

- Un ejemplo que requiere este tipo de solución son las validaciones de usuario, donde lo mejor es desplegar un popup y luego retornar a la pagina donde esta desarrollando la tarea, si el usuario vuelve a la pagina donde se desplegó el popup, se encontrara perdido, ya que no se valido y por lo tanto no pudo terminar la acción que quería ejecutar y la ventana del popup quedo abierta

- La otra situación donde por lo general la mejor solución es un popup es cuando necesitamos ejecutar una tarea que sea ágil y "corta", ya sea completar dos o tres campos agregar un ítem, consultar un dato, etc.

Mecanismo:
La mejor solución para estos problemas es utilizar popup's de aparición transitoria en el frente de la pagina tradicional, bloqueando el acceso al resto de la pagina.
Usualmente, estos popup son parcialmente transparentes, son dibujados sobre la página hasta que explícitamente son cerrados por el usuario.
De esta manera el control de la pagina de no se pierde ya que usuario la única posibilidad que tiene es ejecutar la operación del popup y luego cerrarlo para poder retomar el control de la pagina origen.

Impacto:
A menudo, la Mensaje de Aparición Automática posee un estado volátil, es abierto, usado y cerrado ya sea por que la tarea se completo o bien por la cancelación de la misma, A menudo es programado con pattern "Singleton" [4], esto es solo uno puede ser abierto por vez.

Usualmente los popup son implementados con el tag <DIV>, para asegurarse de que solo puede haber un popup abierto por vez, se crean cuando se invocan y se autodestruyen cuando se cierran.

Cuando un popup es abierto usted necesita controlar manualmente que ya no exista un pop abierto en la misma pagina, alternativamente, esto puede ser simplificado creando el popup al inicio de la sesión y alternar entre mostrarlo y ocultarlo, es mostrado cuando el usuario los necesita y ocultado cuando ya no es necesitado.
Desde el punto de vista del modelo, obviamente el modelo navegacional deberá contemplar, dentro del nodo, la funcionalidad que cumplirá el mismo dentro de la aplicación. Desde el punto de vista de la capa de presentación,

Notas:
A continuación se destacan algunas propiedades que serán de utilidad para el desarrollo del mismo.

“elements” que se pueden usar en hojas de estilo CSS:

Opacity: es usado para producir una apariencia de transparencia

zIndex: es usado para poner el popup al frente de todo, para el resto del documento, por ejemplo Un elemento sólido (opacidad = 1) opacara por completo todo lo demás, además cuando el popup sea abierto `zIndex` será el mas grande.

visibility: puede ser usado para alternar entre mostrar el tag y ocultar el mismo.

Una vez que le damos estilo al popup e identificamos los momentos correctos de apertura y cierre de los mismos, solo nos queda interactuar con el tag <DIV> para lograr representar la apertura y cierre del popup, por lo general el proceso de interacción es similar al de un Portlet – Cada acción del usuario lleva a un XMLHttpRequest

Apariencia del popup:
La apariencia de los popups es una característica que nos permite representar diferentes contenidos sin necesidad de cerrar el documento principal.
Tenemos que balancear entre la facilidad de interacción con el popup, la facilidad de comprensión contra la complejidad de interacción con el resto del documento. El grado de transparencia y el tamaño ayudan a la interacción con le popup, diferentes colores también podrían causar distintos impactos, El mejor color depende de qué contenido se oculta detrás de la aparición del popup

Aspectos que deberemos tener en cuenta:

¿Necesita el usuario ver otra información de la página mientras la mensaje de aparición automática es presente?

> Si, la apertura del popup no debería bloquearme la visión del resto de la página, por eso el documento principal no es opacado completamente.

¿Cuanto tiempo durara abierto el popup?

> Si el popup durara mucho tiempo abierto debemos considerar otro tipo de solución al problema que estamos resolviendo con el mismo

¿Como el usuario abrirá y cerrara el popup?

Hay dos situaciones comunes donde tenemos que abrir un popup

- Al hacer click en un botón o un objeto relacionado al contenido del popup.

- Al posicionarse sobre algo un par de milésimas

A continuación se describen algunas aproximaciones para cerrar el popup:

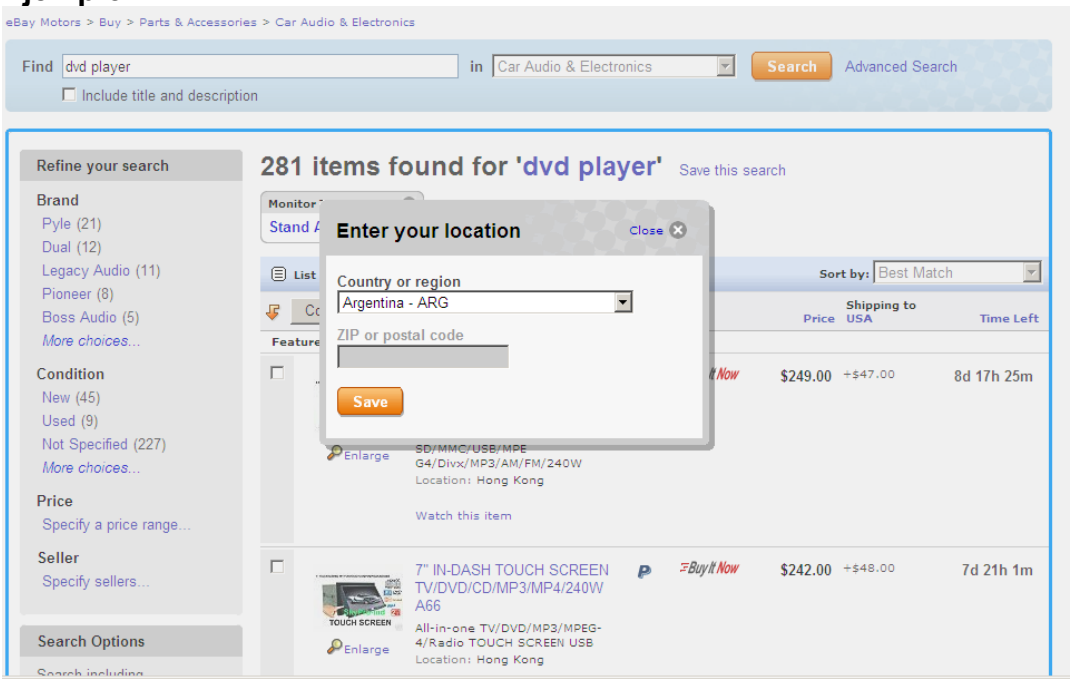
- Cerrar explícitamente con un botón “cerrar”, algunos popups tienen el estilo de “aplicación de escritorio” con una “X” arriba a la derecha del mismo.

- Cerrarlo, como conclusión de la finalización de una tarea sobre el mismo, esto puede ser cuando hacemos un submit de un formulario que contiene el popup, agilizando, ayudando, haciendo más intuitiva la aplicación. esta característica también recae sobre el concepto de feedback, ya que el usuario tiene una respuesta a la acción que realizo.

- Si el popup fue inicializado con un evento “mouse over”, lo cerramos cuando el cursor es llevado fuera del radio del punto donde esta contemplada la funcionalidad “Mouse over”.

- Lo cerramos si paso un cierto tiempo desde la apertura, esto puede ser frustrante ya que el usuario pierde el control de la situación y además no podemos determinar si el usuario se encuentra en el medio de una interacción con el popup. Si usamos esta característica debemos considerar la opción de poder cerrar el popup explícitamente.

Ejemplo:



La imagen muestra como podemos levantar un popup dentro de la aplicación para solicitar un dato mínimo, pero importante, sacando el control a la página principal pero sin perder de vista a que refiere el popup. Una segunda alternativa seria darle un poco mas de trasparencia al popup para poder seguir viendo de alguna manera toda la pagina principal. Este seria ajustando la característica nombrada antes, *opacity*

8. Caso de estudio

8.1. Introducción

Para cerrar el proceso de estudio e integración de conceptos vamos a mirar de cerca un caso concreto de una aplicación que en la actualidad no tiene en cuenta la mayoría de los conceptos mencionados anteriormente.
En la actualidad me encuentro trabajando con una aplicación Web, para una conocida red de clubes (red de gimnasios), que empezó como una PYME y fue creciendo hasta lo que es hoy, la red de gimnasios mas grande de capital federal con una posible expansión a la localidad de la plata y la Santa Fe, los gimnasios están separados por categorías jerárquicas, y cada socio puede entrar a la sucursal, obviamente a las sucursales de igual o menor categoría. Nunca a una sucursal de categoría mayor.
A mediados del año 1997 empezaron con dos sucursales, hoy 11 años después ya son 16 sucursales.
El constante crecimiento del negocio, hizo que el tiempo de vida de las aplicaciones que hoy manejan el negocio acelerara la pendiente de desactualización de las mismas.
La cantidad de datos fue creciendo de manera tal que hoy son 60000 socios activos contra 200 que eran en el año '97.
Hoy por hoy, lo los procedimientos de búsquedas, compra de productos, obtención de la información se ven desbastados frente a la demanda de la gente por ir al gimnasio, al haber crecido tanto la información, y la aplicación haber quedado, tecnológicamente hablando, desactualizado, se hace muy difícil encontrar un dato mas allá de la performance de la aplicación.
Si bien la aplicación es una intranet, no deja de ser una aplicación Web, donde los usuarios de la misma necesitan obtener , procesar, actualizarlos con tiempos de respuesta críticos dada la demanda de los socios ya sea por actualizar alguno de sus datos , pagar cuotas, comprar algún articulo disponible, sacar un carnet, etc. Es decir necesitamos exprimir a fondo la forma en que la aplicación maneja los recursos.

El proceso de refactoring
En los primeros párrafos de esta sección hablamos de cómo iniciar un proceso de migración de una aplicación sin la necesidad de contar con un gran staff de desarrollo.
A continuación analizaremos la ventajas de realizar una migración de a "*pequeños pasos*" - refactorings

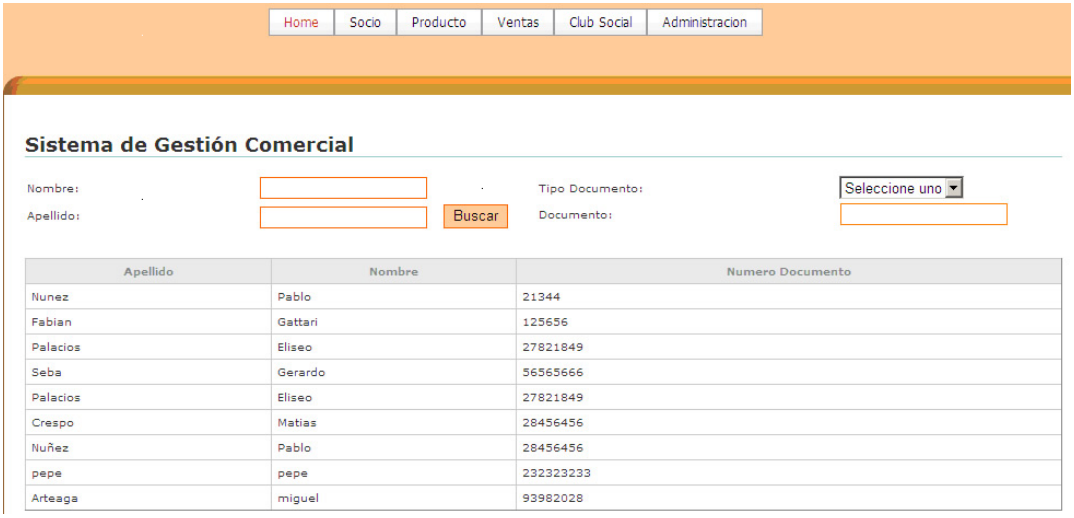
8.2. Agregando AJAX a nuestra aplicación

Motivación: Cada vez que hacemos click en un botón, cuando seleccionamos un ítem de un combo, cuando seleccionamos un ítem de una grilla las paginas son recargadas en su totalidad, esto hace que la aplicación sea lenta, que perdamos el foco en donde estábamos parado, Pero el principal motivo por el cual seria indispensable agregar Ajax es por cuestiones de performance.

Solución propuesta: La aplicación esta programada en ASP.NET, ósea que tendremos que buscar alguno de los Frameworks mencionados que permitan su utilización bajo la plataforma .Net.
Dentro de las mencionadas en la primera sección del estudio, y fácil de integrar podríamos nombrar Magic AJAX, ya que el proceso de integración no requeriría mayor reprogramación, o control de errores.
Silverlight es la posibilidad con mayor cantidad de votos ya que viene con una colección importante de controles o soluciones para poder representar cada una de los refactorings nombrados en la sección anterior, además de la compatibilidad con el entorno de desarrollo Visual Studio .Net.
El proceso de recambio de controles con Microsoft Silverlight puede ser un poco más complicado que si usamos Majic AJAX.
Este último no posee controles AJAX, sino que trabaja con regiones y los controles que pongamos dentro de estas regiones no recargaran la totalidad de la página sino que recargaran los controles que involucren algún cambio sobre la página.
Ósea que deberemos evaluar ventajas y desventajas de utilizar cada una de las soluciones propuestas, si bien el objetivo de ambas es el mismo, Microsoft Silverlight requeriría mayor tiempo para llevar a cabo el refactoring sobre la aplicación, ya que si como tenemos controles heredados, deberíamos reescribirlos con los que proveen la funcionalidad AJAX.

8.3. Refinación de Búsquedas:

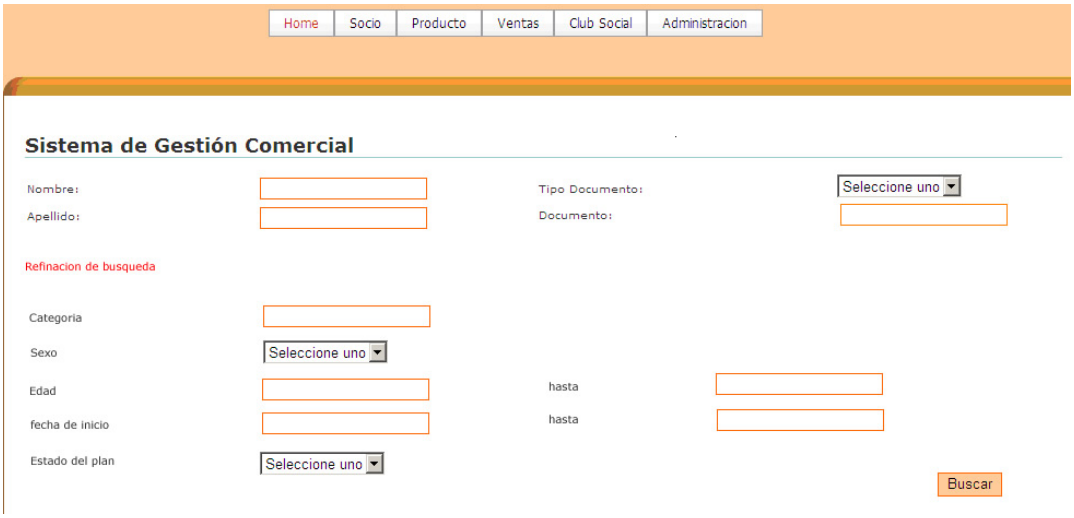
Motivación:
La base de datos de socios del club contiene alrededor de 60000 socios, a menudo los usuarios del sistema pierden grandes cantidades de tiempo para poder localizar a un socio.
La falta de validaciones y mal diseño de la aplicación lleva a tener cargados datos repetidos, eso hace que las búsquedas den como resultado grandes recordsets's donde nos obliga a terminar de realizar la búsqueda manualmente.
En la actualidad Hay muchos datos inválidos en el sistema, datos de socios repetidos, números de documentos repetidos, etc.
Indefectiblemente si hiciera una búsqueda en esta pagina por el nombre "*Roberto*" me traería una lista infinita de de registros, casi imposible de manejar en la pagina.
Hoy por hoy la búsqueda se puede realizar por un par de campos (nombre, apellido, número de socio, número de documento, fecha de ingreso).
En la sección anterior definimos como realizar el proceso de refinación de búsqueda, para solucionar este problema seria ideal agregar filtros para poder acotar el resultado devuelto en la búsqueda inicial con pocos parámetros.



Mecanismo:
En la sección anterior se estudio el proceso para solucionar el amplio espectro de registros y los problemas que existen para poder identificar a un registro en particular en una larga lista, con múltiples paginas como resultado.
Si bien esta aplicación no es no es una aplicación de e-comerse, encontrar un socio rápido es importante, para evitar que el mismo tenga que esperar cada vez que se acerque al mostrador para realizar una compra.

Impacto:
Si bien el sistema actual no tiene un modelo OOHDM, esta diseñado con un modelo de múltiples capas, y con programación orientada a objetos.
Lo ideal seria refactorizar la capa de presentación y agregar un panel con más opciones para poder acotar el resultado de la búsqueda.
Algunos de los campos que podrían ayudar a refinar la búsqueda podrían ser la categoría de socio, la sucursal donde tiene comprado el plan, edad, sexo, fecha de inicio, estado del mismo, entre otros.
Una opción es componer este refactoring con el refactoring "agregar información", si agregamos información a los ítems del resultado también ayudaría a identificar unívocamente un socio, en el agregado de información destacamos que un dato importante podría ser la foto del socio.

Ejemplo:



8.4. Enriquecimiento de índice:

Motivación*:* Al recuperar la información de los socios en un proceso de búsqueda, la información que muestra la aplicación muchas veces suele ser pobre, y no suficiente para tomar una decisión para saber de que socio estamos hablando, para conocer algún otro tipo de información necesitamos ingresar a ver su ficha, esto hace que la mayoría de las veces ingresemos a la ficha de un socio y no era el que buscábamos.
Esto hace que la mayoría de las veces ingresemos a ver información de un socio que no es el que estábamos buscando.

Mecanismo*:* Si bien el problema que queremos solucionar no es literalmente un índice, intenta cumplir la función, en la búsqueda de socios queremos poder encontrarlo mas rápidamente, y cuanta mas información mostremos, mas fácil nos resultara la tarea. Para ello vamos a utilizar el proceso de enriquecimiento de índices que explicamos en la sección anterior con las propiedades que el negocio implica.

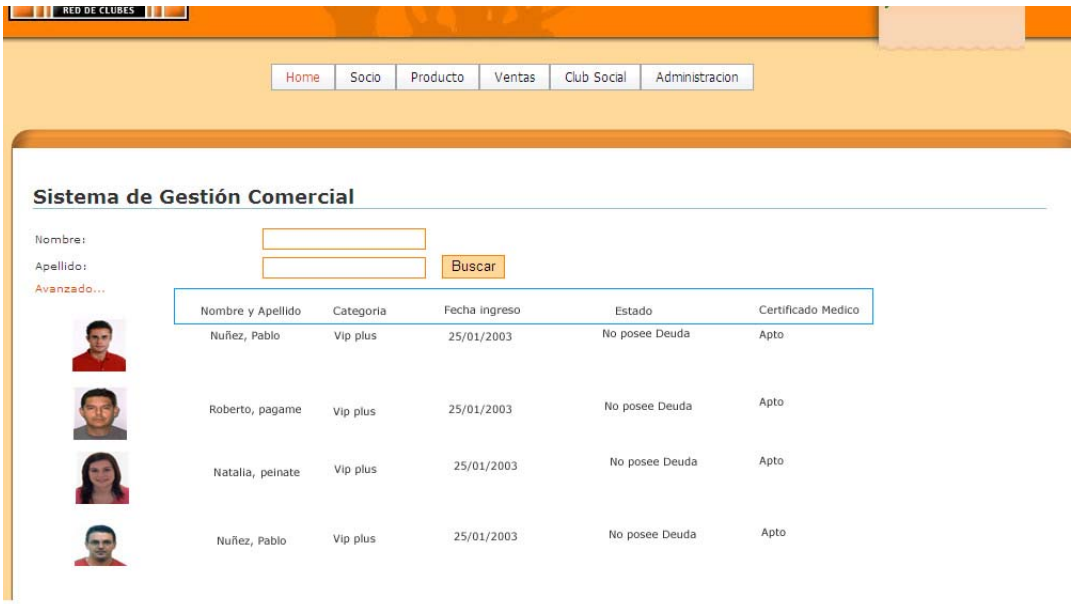
Impacto:
Para Solucionar este problema vamos a utilizar el procedimiento que describimos mas arriba para enriquecer el "índice" de la información de socios.

Paso 1(Agregar Información)

Lo primero que vamos a hacer es enriquecer la información que se muestra al realizar la búsqueda de Para solucionar este tipo de problemas lo ideal seria agregar información a cada uno de los registros que vamos a mostrar, entre la información que seria importante anticipar podemos destacar:

- La categoría del plan que tiene comprado
- Si tiene deuda
- La fecha de nacimiento
- El año de ingreso al club
- Estado del certificado medico (es obligatorio que este apto para realizar actividad física)
- La foto la foto que posee el socio en el carnet, este dato no ayudara a identificar

Para esto podríamos usar cualquiera de las técnicas antes mencionadas para poder anticiparle al usuario mayor información acerca del socio sobre el cual se encuentra parado.
Conociendo los requerimientos de los usuarios de la aplicación, Yo me inclinaría por la opción de un popup con la información adicional que indique la información que indicamos arriba.



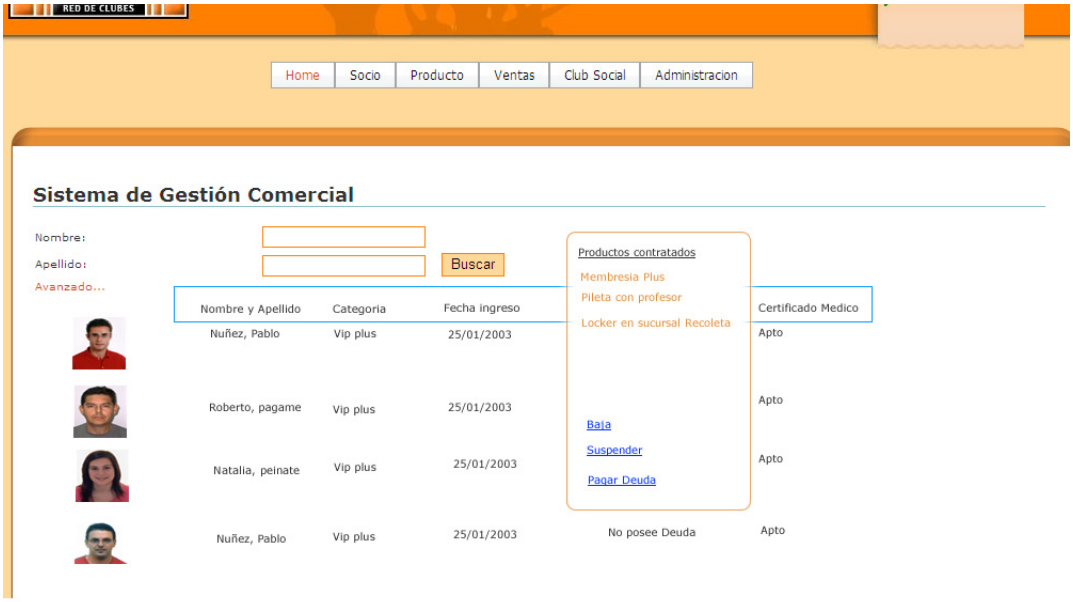
La imagen muestra como seria una aproximación de cómo quedaría el índice de socios con el agregado de algunos campos mas a la pagina.

Pasó 2 (Agregar operación):

En la actualidad para poder realizar algún tipo de operación sobre la información del socio debemos ingresar al portal del mismo.
Hay operaciones que seria ideal que pudiéramos hacer sin la necesidad de tener que ingresar a su portal.
Dentro de esas operaciones esenciales podríamos destacar:

- Pagar la deuda pendiente (si tuviese)
- Inactivar socio (suspender)
- Dar de baja socio
- Cargar certificado medico

Ejemplo:


RED DE CLUBES
Home
Socio
Producto
Ventas
Club Social
Administracion
Sistema de Gestión Comercial
Nombre:
Apellido:
Avanzado...
Buscar
Nombre y Apellido
Categoria
Fecha ingreso
Certificado Medico
Nuñez, Pablo
Vip plus
25/01/2003
Apto
Roberto, pagame
Vip plus
25/01/2003
Apto
Natalia, peinate
Vip plus
25/01/2003
Apto
Nuñez, Pablo
Vip plus
25/01/2003
No posee Deuda
Apto
Productos contratados
Membresia Plus
Pileta con profesor
Locker en sucursal Recoleta
Baja
Suspender
Pagar Deuda

***Drag & Drop**:*

Motivación: Esta Red de clubes no solo vende servicios, sino que hay una gran variedad de artículos relacionados al negocio entre los cuales podríamos nombrar:

- Anteojos de sol
- Antiparras para natación
- Ganadores de peso
- Creatina
- Gorros de natación
- Patas de rana

Hoy por hoy no existe un carrito de compras en la aplicación, lo ideal seria poder tenerlo para poder seleccionar los productos que podría comprar el socio y luego poder efectivizar la misma, esto agilizaría por completo el proceso de compra ya que en la actualidad solo existe la posibilidad de comprar de a un solo producto por vez, donde se le solicita al cliente los datos de pago por cada vez que debemos cobrarle.
Todo esto es consecuencia del mismo crecimiento que nombramos en la introducción del caso de estudio. Cuando la red comenzó su actividad, solo vendía actividades, el crecimiento de la actividad misma como proceso de cura o prevención de enfermedades fue uno de los principales causantes del crecimiento de la actividad

Mecanismo:

Para este problema podemos analizar diferentes propuestas:

1. Agregando operación al índice de productos
2. tener una sección con una cesta, y permitir arrastrar los productos hasta el mismo.

Nosotros creemos que la mejor solución es la que propusimos en la sección de refactorings donde indicamos una solución al carrito de compras la cual concuerda con la segunda opción propuesta.
Deberíamos destinar una región común con forma de cesta de compras (al estilo supermercado) donde el usuario de la aplicación pueda arrastrar

Impacto: si bien existen varias maneras de implementar un carrito de compras, en la sección anterior propusimos varias soluciones para solucionar este faltante.
Un ventaja para la implementación es que la aplicación esta programada orientada a objetos, por lo que deberemos refactorizar la interfaz grafica para poder relacionar el carrito de compras con los productos que se venden en las sucursales de la red.

En la sección anterior definimos cuales serian los pasos y propiedades que deberíamos tener en cuanta al momento de decidir darle a la aplicación un comportamiento de drag & drop

El evento Drag & Drop en si, se componen de una serie de eventos que son los que realmente tienen la logica para llevar a cabo el proceso, a continuación detallamos los distintos eventos que se ejecutan desde que elegimos el elemento draggable hasta que los depositamos en la region que elegimos.

- **Generación de la página**: este evento deberá representar gráficamente cada una de las instancias del índice de artículos a la venta, debemos tener en cuanta que esta representación no debería ser un conjunto de controles sueltos, sino que debería ser un modelo grafico (oohdm) definido para la representación de los artículos.

- **Posicionar el mouse sobre objeto**: a cada una de las representaciones graficas deberemos darle la inteligencia necesaria para saber que ese objeto es “draggable” o no.

- **Drag iniciado:** al presionar el mouse sobre el objeto y moverlo debemos marcar el objeto como que esta en estado de movimiento, esto es cambiar la representación del mouse o indicarlo de alguna otra manera como cambiar el color del borde del articulo que arrastramos.

- **Drop aceptado / rechazado**: así como definimos la representación grafica de los elementos que podemos arrastrar, debemos definir la representación grafica del carrito en si mismo, ósea la región donde vamos a hacer el “drop”. Si el drop se hiciera sobre cualquier otra región de la página, no seria valido.

- **Drop vuelve a la posición original**: si el drop fuese rechazado debemos dejar todo como estaba hasta antes de iniciar el drag. El objeto arrastrado, vuelve a la posición original, en realidad nunca dejo de irse de esta pero, si su representación grafica, vuelvo a cambiar la representación del puntero de Mouse, vuelvo la pantalla a la posición original como si nada hubiese pasado para poder seguir con el normal desarrollo de acciones.

Otro aspecto importante a tener en cuenta son los actores que participan en la funcionalidad de Drag & Drop de la pagina.

- **Cursor:** si hacemos mousedown sobre una región de la pagina que acepte Drag & Drop debemos indicárselo al usuario, como así también debemos indicarle que en la región donde se hizo mousedown no se puede realizar Drag & Drop. Esta indicación se realiza mediante el cambio de representación del puntero. también mediante este deberemos indicarle al usuario la región permitida para poder hacer el drag. Para poder hacer el mouseup

- **Representación del Modulo**: debemos elegir la mejor representación para el modulo sobre el cual vamos a hacer Drag & Drop, la representación grafica del objeto (capa de presentación) es que deberá indicar al mouse que estamos en un elemento que podemos arrastrar.

- **Ubicación original del modulo sobre el cual se hace Drag:** debemos saber cual es la posición original del elemento que arrastramos hasta el momento que hacemos mouseup, si lo hacemos sobre una posición valida, reiniciamos todo y movemos el objeto que estábamos arrastrando hacia la posición o contenedor donde soltamos el Mouse, sino volvemos el elemento arrastrado a la posición original , y reseteamos la variable sobre la cual habíamos almacenado los puntos y dejamos todo como estaba hasta el instante anterior al mousedown.

- **Posición sobre la cual hicimos drop:** cuando el arrastrado fue correcto debemos actualizar el estado de la pantalla, debemos actualizar (el target) desde donde iniciamos el evento, como el movimiento fue un drop, debemos actualizar el stock o bien quitar el elemento de la lista de módulos.

En la actualidad hay varia librerías para implementar la funcionalidad pero la mayoría de ellas con templan un mismo patron que tienen en cuenta los mismos criterios al momento de implementar el diseño de una funcionalidad de Drag & Drop.

9. Conclusiones

En el inicio del trabajo de grado estudiamos las características de las aplicaciones Web tradicionales, con ello vimos que la mayoría de estas eran falencias que poseen las mismas.

Un tiempo mas tarde estudiamos las características de las Aplicaciones Ricas en Internet, estudiamos su arquitectura, como difiere un request en una aplicación rica y en una aplicación tradicional, también vimos como evitar las sucesivas recargas de las páginas, las diferencias de performance entre unas y otras.

En el tramo final analizamos el concepto de refactoring, y como podemos aplicarlo a las aplicaciones Web, propusimos una serie de refactoring para poder mejorar ciertos aspectos de una aplicación Web tradicional y acercar su performance a las aplicaciones ricas.
Teniendo en cuenta estos podemos solucionar varios inconvenientes que poseen los usuarios al utilizar una aplicación:

- Interactividad: agregándole feed back a la aplicación podemos indicar al usuario el resultado de las acciones, mostrarle alertas, para que el usuario este en todo momento alertado de que es lo que pasa dentro de la aplicación.

- Confianza: proveer al usuario cierta sensación de seguridad con cada acción que ejecuta dentro de la aplicación, no dejamos al asar operaciones, el usuario ve resultados concretos al hacer un click.

- Aplicaciones mas parecidas a las de escritorio: con las aplicaciones Ricas en Internet podemos darle al usuario ciertos comportamientos con no podíamos con las aplicaciones Web tradicionales, además Utilizando Hojas de estilo, skins, y herramientas de diseño podemos darle un look and feel mas parecido al de una aplicación de escritorio,

- Performance: ya no son tantos los request que se envían al servidor, solo requerimos datos cuando los necesitamos o bien muchas veces hacemos la carga de datos anticipadamente

- Mas Información: debemos enriquecer los datos que mostramos, para poder identificar mas rápidamente lo que buscamos, uno de nuestros primeros refactorings propuestos fue el “agregar a información”, para poder enriquecer la aplicación.

- Recargas de página: las sucesivas recargas de página son uno de los principales problemas de las aplicaciones Web, aplicar un Framework Ajax a la aplicación podría empezar a solucionarnos varios inconvenientes, entre ellos las sucesivas recargas de página.

- Trafico de red: al no tener tantos request hacia el servidor nuestra aplicación no ocupa tanto trafico de red, por lo tanto vemos que con cada pequeño cambio que vamos realizando, vamos ganando en otras cosas, en este caso utilizar Ajax y gestionar requerimientos al servidor de manera controlada, hace que disminuyamos el trafico de red.

- Problemas navegaciones: analizamos la problemática que existe como consecuencia de la falta de información, de malos diseños, llegamos a la conclusión que muchas de las activaciones a través de links nos llevaban a un target que no contenía la que estábamos buscando. Aplicando refactorings al modelo navegacional podemos solucionar muchos de estos problemas.

La técnica de refactorizacion es una técnica que se utiliza en aplicaciones orientadas a objetos, nosotros llegamos a la conclusión que podemos llevar nuestra aplicación Web hacia una aplicación “rica”, haciendo pequeños cambios, ósea, haciendo sucesivos refactoring sobre las funcionalidades, sobre la presentación, sobre la navegación de la aplicación.

Lo ideal es focalizar la funcionalidad y/o aspecto de la aplicación que queremos cambiar e ir introduciendo los refactoring’s uno por uno, esto es, refactorizamos una parte pequeña de la aplicación, vemos que no hayamos introducido errores, y una vez que el usuario se familiarice con el mismo volvemos a empezar con otra parte de la aplicación.

De esta manera, el usuario no se sentirá perdido en la aplicación, vera que una funcionalidad cambio, luego de su acostumbramiento, vera otro cambio, y luego otro, y así sucesivamente.

Si el cambio fuese hacia una aplicación totalmente nueva, el usuario puede verse perdido, y resultarle difícil la adaptación.

10. Trabajos Futuros

Vimos que la mayoría de los refactoring involucraban cambios en más de una capa del modelo de la aplicación Web.
En la actualidad se esta trabajando para poder tener una herramienta de diseño Web que pueda contemplar un cambio en el modelo navegacional, por ejemplo, o en el modelo de aplicación y mapear esos cambios en el modelo navegacional y/o a nivel de presentación si el refactoring lo requiere.

Una herramienta para refactorizar aplicaciones Web deberá tener la inteligencia suficiente para sincronizar los cambios que produzca un refactoring e impactarlos en las capas subsiguientes, esto es, si el refactoring es a nivel navegacional y conlleva cambios en la capa de presentación o bien sobre la capa de aplicación y viceversa.

Bibliografía

[1] http://labs.adobe.com/wiki/index.php/AIR

[2] http://ajaxpatterns.org

[3] D. Van Duyne, J. Landay, J. Hong. *The Design of Sites*. Addison-Wesley 2003.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns. Elements of reusable object-oriented software*, Addison Wesley 1995.

[5] Addison Wesley - Refactoring Improving the Design of Existing Code - Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts

[6] http://www.magicajax.net/

[7] Y.Ping and K.Kontogiannis. Refactoring Web sites to the Controller-Centric Architecture. In *Proc. of the European Conf. on Software Maintenance and Reengineering (CSMR'04).* Tampere, Finland, 2004.

[8] Refactoring to Patterns Joshua Kerievsky

[9] G.Rossi, D.Schwabe, A.Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of Hypertext'97*. Southampton, UK, 1997.

[10] www.campusred.net/TELOS/articuloperspectiva.asp?idarticulo=2&rev=73

[11] http://openrico.org/rico/home.page

[13] http://msdn.microsoft.com/msdnmag/issues/07/06/Silverlight/default.aspx

[14] P. Van Gorp, H. Stenten, T. Mens, S. Demeyer. Towards automating source-consistent UML Refactorings. In *Proceedings of the 6th Int. Conference on UML*, 2003.

[15] http://es.15.org/wiki/Rich_Internet_Application

[16] http://es.15.org/wiki/Web_2.0

[17] Web Patterns. http://17terns.org/

[18] Web Design Patterns. http://www.18.com/ patterns/

[19] Schwabe, D., Rossi, G. (1998) An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems 4(4), Wiley and Sons, 1998.

[20] M. 20, J. Nanard, P.Kahn. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In *Proc. of Hypertext'98*. Pittsburgh, USA. 1998.

[21] Google Web Toolkit http://code.google.com/webtoolkit/

[22] Rich Internet Applications: Design, Measurement, and Management Challenges, Chris Loosley Senior Director, SLM Technologies Keynote Systems, 2006

[23] Web Design Patterns. http://www.welie.com/ patterns/

[24] Web Patterns. http://webpatterns.org.