Especificación de Interfaces y Patrones RIA

Ingeniero en Sistemas de Información Diego Oscar Campana

Director: Doctor Gustavo Rossi

Tesis presentada a la Facultad de Informática de la Universidad de La Plata como parte de los requisitos para la obtención del título Magíster en

Ingeniería de Software.

La Plata, Marzo 2009

Facultad de Informática

Universidad Nacional de La Plata

República Argentina

A mis hijos Mateo y Renzo por su invalorable tiempo.

A mi mujer Liz por su paciencia y amor.

Agradecimientos

Un agradecimiento especial al Doctor Gustavo Rossi por aceptar la dirección de esta tesis y guiarla durante su realización.

Un sincero agradecimiento a Matias Urbieta por su generoso aporte de conocimientos y sus enseñanzas.

Un enorme gracias a mis padres por el incondicional apoyo durante toda mi carrera.

Contenido

Capítulo 1. Introducción y Objetivo

1.1 Introducción

Desde los inicios de Internet las aplicaciones Web tradicionales centran su actividad en una arquitectura Cliente – Servidor donde todo el procesamiento es realizado en el servidor y el cliente es utilizado para mostrar contenido estático.

La mayor desventaja con este sistema es que toda Interacción con la aplicación debe pasar por el servidor.

Haciendo uso de tecnologías que puedan ejecutar instrucciones en la máquina del cliente, las aplicaciones “ricas” (RIA) pueden eludir este bucle lento y sincrónico para muchas de las iteraciones del usuario.

Los estándares de Internet han evolucionado en forma lenta y continua acomodándose a estas técnicas, por lo cual es difícil trazar una línea entre lo que constituye una aplicación RIA y lo que no. Pero todas las aplicaciones RIA comparten una característica, introducen una capa intermedia, comúnmente llamada “motor cliente” (client engine), entre el cliente y el servidor, responsable de dibujar la interfaz de usuario y de la comunicación con el servidor, siempre de ser posible en forma asincrónica.

La figura 1 muestra las capas que integran la arquitectura de una aplicación RIA, donde podemos ver el “contenedor de ejecución de interface rica” comúnmente llamado “motor cliente”.

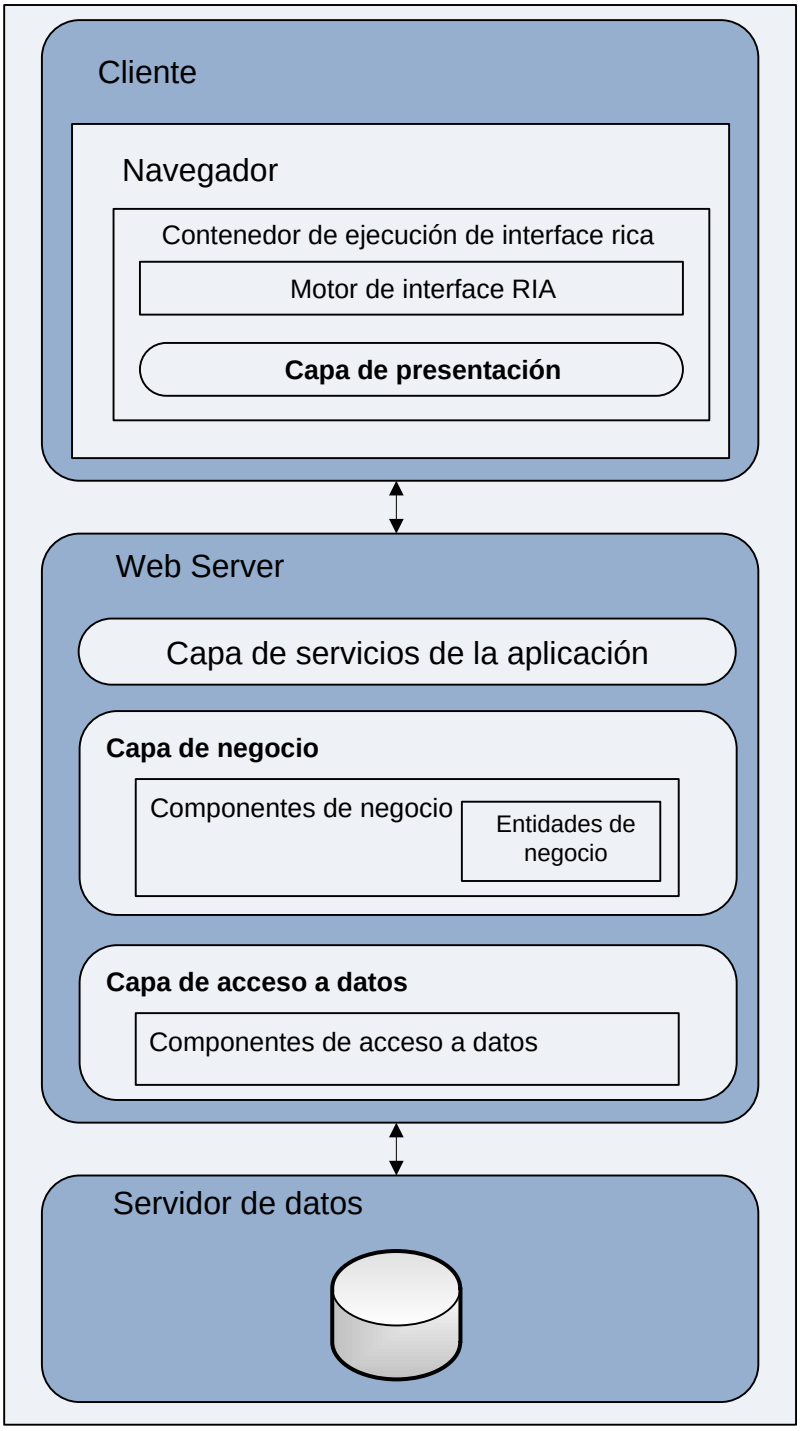


Figura 1: Arquitectura de las aplicaciones RIA [23].

Existe actualmente un gran número de comportamientos RIA identificados como patrones Web, los cuales atienden a diferentes necesidades y soluciones: búsqueda, selección, lectura, navegación, etc. mejorando la funcionalidad provista en las aplicaciones Web tradicionales.

Estos patrones Web mejoran la funcionalidad provista en las aplicaciones Web tradicionales, haciendo uso de la tecnología RIA.

Si bien las aplicaciones RIA pueden mejorar en mucho la experiencia del usuario, requiere una gran tarea de análisis y conocimiento de la misma, como también de los usuarios que la utilizan, para poder realmente agregar valor a su uso y funcionalidad, no siendo sólo tecnología aplicada a la apariencia.

Para poder crear una aplicación RIA o transformar una aplicación Web tradicional en una aplicación RIA existen muchas tareas a cumplir: es necesario hacer uso de técnicas y herramientas para modelar la misma, diseñar o modificar su arquitectura, seleccionar las tecnologías a utilizar de acuerdo a las prestaciones y necesidades, realizar pruebas del nuevo funcionamiento, etc.

1.2 Objetivo

El propósito de este trabajo es mostrar y proponer un camino posible para la transformación de una aplicación Web tradicional a RIA, haciendo uso de técnicas y herramientas para especificar el viejo y nuevo estado de la misma, dando una completa visión de los cambios arquitecturales y adaptaciones en las aplicaciones desarrolladas, con y sin patrones de diseño.

1.3 Contribuciones

Del cumplimiento de los objetivos de este trabajo se desprenden las siguientes contribuciones:

- Mostrar el estado actual de las tecnologías utilizadas en aplicaciones RIA. Sus ventajas y casos de aplicación con ejemplos.
- Tener una visión y conocimiento de las herramientas y lenguajes para especificar aplicaciones RIA. Diagramas de arquitectura, patrones de diseño, comportamiento, secuencia, ADVs y ADVs State Charts, etc.
- Evaluar las ventajas de disponer de una aplicación correctamente diseñada haciendo uso de patrones de diseño para facilitar el cambio a RIA.

- Explicar una propuesta que mejore las especificaciones de patrones Web actuales, incluyendo en las mismas ADVs y ADV-Charts.
- Mostrar los cambios que surgen en los Tests de la aplicación y las herramientas que nos ayudan a verificar el funcionamiento de una aplicación RIA.
- Conocer el estado actual del conocimiento en trabajos relacionados sobre el uso y aplicación de RIA.

1.4 Alcance

Este trabajo tiene como alcance mostrar el estado actual de las aplicaciones RIA, sus ventajas y aportes a la usabilidad de la Web. Sin entrar en detalle de las tecnologías mencionadas dará una visión general de la mismas. Se mostrará el uso de técnicas y lenguajes de especificación de suma utilidad e importancia para el proceso de creación o transformación a RIA, proponiendo alternativas a los tipos de especificaciones encontrados en sitios y bibliografía actual.

1.5 Organización del texto

Esta tesis se estructura en ocho capítulos que se describen a continuación:
Capítulo 1: Expone una introducción global, el objetivo y el alcance del trabajo.
Capítulo 2: Presenta un resumen detallado de las diferencias entre las aplicaciones Web tradicionales y las aplicaciones RIA. Sus limitaciones, ventajas y la arquitectura que las soporta.
Capítulo 3: Explica cómo funcionan diversas tecnologías utilizadas para construir aplicaciones RIA, complementarias muchas veces. Aspectos técnicos y ventajas de las mismas.
Capítulo 4: Describe la composición de tareas y etapas en OOHDM, como metodología y herramienta para el análisis y especificación de aplicaciones RIA, enfocándonos en la capa de interface abstracta mediante el uso de ADV-Charts, que usaremos en el siguiente capítulo.

Capítulo 5: Muestra cómo analizar y especificar diferentes patrones Web en una aplicación RIA, proponiendo completar los tipos de especificaciones que generalmente se realizan y usan, adicionando dos secciones: ADV y ADV-Chart, dando ejemplos gráficos de los mismos.

Capítulo 6: Describe un camino posible para la transformación de una aplicación Web tradicional a RIA. Mostrando diferentes diagramas y analizando los cambios en el código de una manera simple.

Capítulo 7: Analiza la transformación del capítulo 6 a partir de una aplicación Web creada utilizando patrones de diseño conocidos. Menciona y da ejemplos de modificaciones en los test de la aplicación luego de la transformación.

Capítulo 8: Realiza una reseña de trabajos relacionados, muchos de los cuales sirvieron de base para la elaboración de este trabajo.

Posteriormente se describen los pasos a seguir para transformar una aplicación Web tradicional a RIA.

Por último se detallan las conclusiones derivadas de la investigación teórica y de los ejemplos prácticos desarrollados.

Capítulo 2. Aplicaciones Web

2.1 Tradicionales

2.1.1 Definición

Las aplicaciones Web tradicionales son aquellas que presentan documentos HTML planos en un navegador y reciben de vuelta peticiones HTTP.

En las aplicaciones Web de tipo cliente delgado (thin-client) no existe código del lado del cliente: javascript, applets, etc. Solamente HTML. El navegador es tratado como si fuera un terminal, con dos notables excepciones. (WebApplications) [38]

El cliente puede guardar un estado (stateful), por ejemplo los datos pueden ser mantenidos en el cliente a través de tres métodos, reescritura de la URL, campos ocultos en un formulario HTML y usando cookies.

El cliente también puede cambiar el estado actual utilizando opciones del navegador, por ejemplo el botón de ir hacia atrás.

Todas las aplicaciones Web son guiadas por eventos. El servidor recibe peticiones HTTP desde un cliente. Estas peticiones HTTP son eventos. Específicamente, un evento Get o Post. Para nuestro propósito ambos pueden considerarse peticiones HTTP genéricas [12].

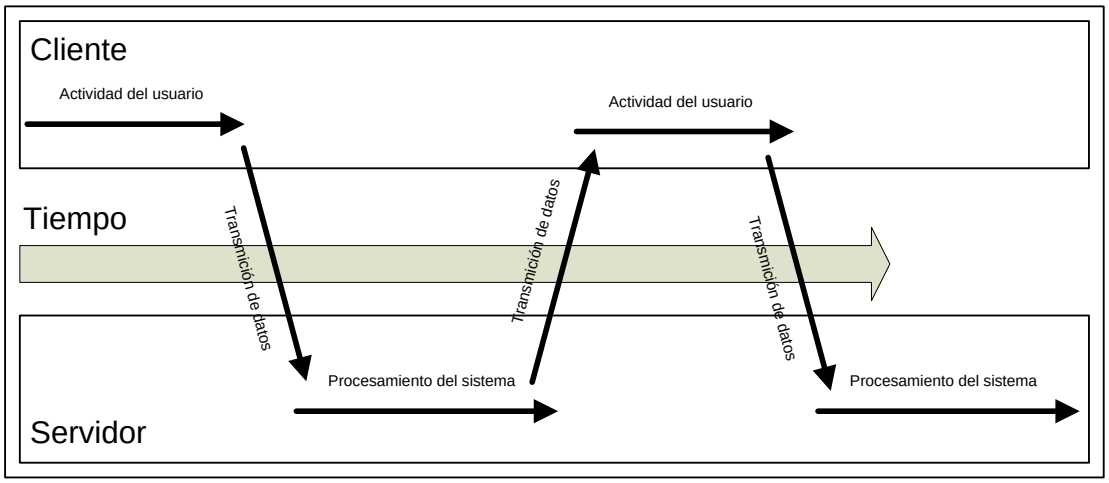


Figura 2: Procesamiento de datos en aplicaciones Web Tradicionales.

La figura 2 nos muestra la interacción entre un cliente y un servidor, mediante el envío de peticiones HTTP y su posterior respuesta, repitiéndose la secuencia en el tiempo.

2.1.2 Capa de presentación

En las aplicaciones Web, el código del lado del servidor que es responsable de la interfaz de usuario es usualmente denominado como capa de presentación, esto quizás es para distinguirlo del navegador cliente el cual está más cerca del usuario y generalmente denominado interface de usuario.
En otras palabras, la capa de presentación es la capa que construye la interface de usuario pero reside en el servidor.

2.1.2.1 Evaluación de eventos

Cuando el servidor de la aplicación Web recibe un evento, debe evaluar el mismo y decidir cómo responder al cliente.
Este proceso de evaluación puede incluir interacción con la lógica de negocio de la aplicación. Esta lógica de negocio puede ser implementada como una capa de negocio o modelo.
Esta capa de negocio generalmente tiene que ser persistida. La persistencia en la mayoría de los casos es realizada mediante una base de datos.
Esta separación de la capa de presentación, la capa de negocio y la capa de persistencia es a menudo llamada en la literatura de las aplicaciones Web como el modelo de 3 capas (3-tier model). Diferente de las aplicaciones cliente/servidor tradicionales las cuales son llamadas generalmente de 2 capas (2-tier).
Las aplicaciones de 3 capas son más flexibles. Físicamente todas las aplicaciones de 3 capas pueden ser implementadas en diferentes servidores, esto puede ser una ventaja para una mayor escalabilidad y parael procesamiento deun número mayor de peticiones de clientes en forma simultánea.

2.1.2.2 Páginas HTML

La capa de presentación es también responsable de enviar las páginas HTML de salida al cliente. Cada vez que se envía al cliente una nueva página (o marco) de HTML, se puede considerar que el cliente cambió de estado. El navegador está mostrando una nueva página.
Con el fin de construir una salida HTML apropiada, la capa de presentación puede necesitar acceder a información del negocio, como por ejemplo una lista de facturas pendientes de salida. Esto puede ser obtenido enviando un mensaje a la capa de negocio y solicitando una lista de tales facturas a la base de datos. La capa de presentación entonces podría construir una tabla HTML para cargar con los detalles de las facturas y enviarla hacia el cliente para ser mostrada en el navegador.

2.1.2.3 Interacción entre el cliente y la capa de presentación

Si observamos únicamente la interacción entre el cliente y la capa de presentación del servidor de la aplicación, podemos ver que el cliente envía una serie de peticiones HTTP como entrada al servidor de aplicación. Esto se puede considerar como un evento Get o Post en un nivel físico.
Como muestra la figura 3, el servidor de aplicación procesa las peticiones HTTP y retorna al cliente una serie de datos HTML para construir la página o marco requerido.

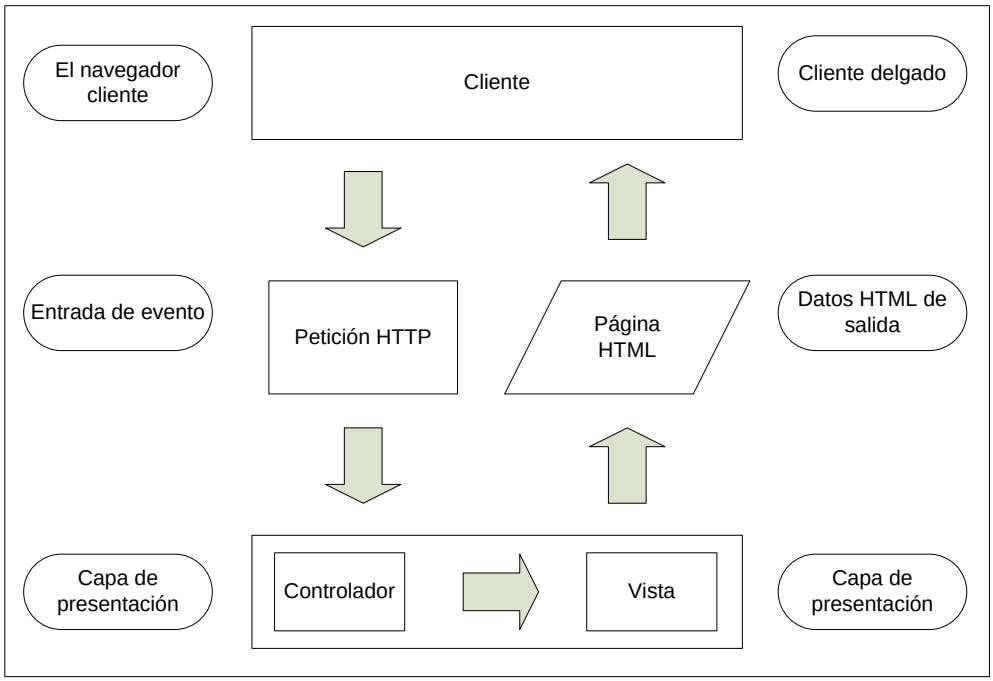


Figura 3: Interacción con la capa de presentación.

En estos eventos físicos se puede agregar una semántica lógica a los datos que acompañan el evento.

Por ejemplo, si un "submit" es presionado, un parámetro adicional "evento=submit" podría ser agregado en los datos de la petición HTTP. Esto puede utilizarse para transformar el evento físico Post en un evento lógico "submit".

Este mecanismo es usado ampliamente en las aplicaciones Web para identificar mensajes específicos del cliente al servidor.

Actualmente lo que sucede es más complejo. Un cliente en un estado dado, tal vez produce un evento, el cual se acompaña de ciertos parámetros:

[Estado actual + un evento + algunas condiciones] → [Las acciones + estado siguiente]

En la recepción del evento, ciertas condiciones son evaluadas, las cuales podrían requerir los parámetros del evento. La combinación del estado actual, el evento, y la evaluación de las condiciones determinará el siguiente estado y posiblemente algunas acciones que deben realizarse cuando la transición de estado es hecha. (Anderson, 2000)

2.1.3 Arquitectura MVC

Existe una arquitectura bien establecida para las interfaces de usuario que facilitan los sistemas guiados por eventos conocida como Modelo, Vista y Controlador (MVC).

Existen dos requerimientos para el controlador en la arquitectura MVC. El controlador debe determinar el control del flujo del sistema. Debe procesar un evento entrante de la interface de usuario y determinar el siguiente estado de la interface de usuario. El segundo requerimiento es enviar un mensaje a la capa del modelo (lógica de negocio) indicando las acciones requeridas. Esto puede cambiar el estado de la capa de negocio subyacente.

En la figura 4 se puede ver la inclusión del modelo en el diagrama de interacción de capas

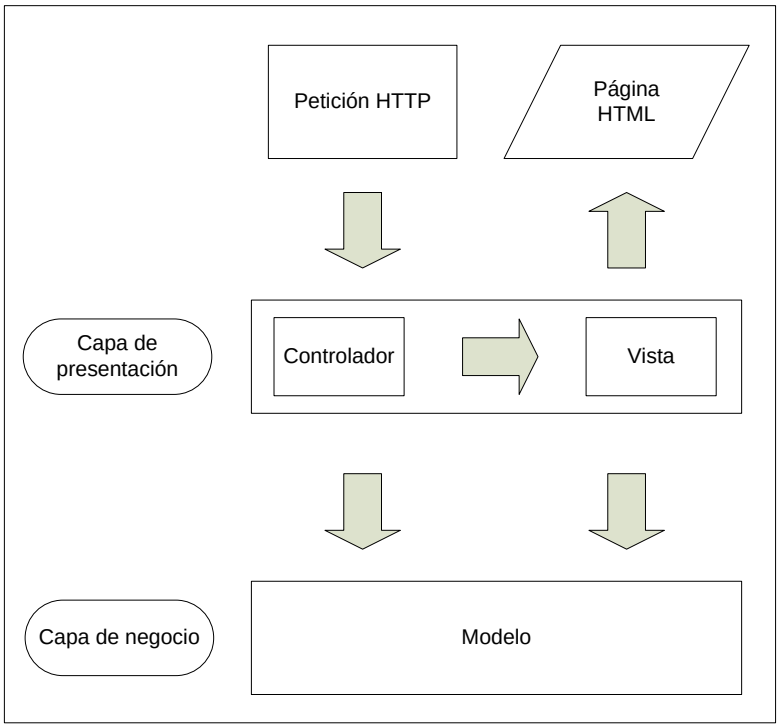


Figura 4: Arquitectura MVC simplificada.

Debe quedar claro que el estado de la capa de negocio y el estado de la interface son dos cosas diferentes, aunque a menudo relacionadas.

En un ambiente GUI el modelo es responsable de notificar a las clases de la vista 3que ha ocurrido un cambio. Normalmente, con un cliente delgado, no hay manera de notificar a la vista existente en el cliente.

De esta manera, el funcionamiento de la arquitectura MVC, es más simple en un cliente delgado que en un entorno GUI.

En este funcionamiento simplificado, el controlador procesa la entrada, toma los eventos, considera el estado actual del cliente y determina el siguiente estado de la interface de usuario. Basado en el siguiente (nuevo) estado determinado, el controlador invoca la vista apropiada que muestra la salida requerida.

Aquí se enfocan las responsabilidades del controlador, para regular el flujo y estado de la interface de usuario, a diferencia del estado del sistema del negocio subyacente.

Como se construye cada vista en demanda, deben requerirse datos del modelo cada vez que se instancia una vista. De este modo, no hay notificaciones del modelo a la vista.

Las vistas y los controladores pueden ser considerados la capa de presentación en una aplicación Web.

El modelo por otro lado, está separado de los controladores y vistas de la capa de presentación. Su función es proveer servicios del negocio a la capa de presentación incluyendo acceso al almacenamiento persistente. [1]

Tanto el controlador como la vista envían mensajes hacia el modelo, donde para acciones determinadas obtendrán respuesta.

2.1.4 Características y limitaciones

2.1.4.1 Características

Click – Esperar – Refrescar: cada interacción del usuario con la aplicación cumple este ciclo, el usuario hace **click** en un enlace, botón, etc. produciendo el envío al servidor de una petición. El usuario **espera** la respuesta del servidor, que al llegar **refresca** al navegador con una nueva página o marco HTML.

Comunicación sincrónica: físicamente la comunicación entre el cliente y el servidor es sincrónica.

En la figura 5 se muestra como el flujo de trabajo en las aplicaciones Web tradicionales, es guiado por la navegación entre páginas.

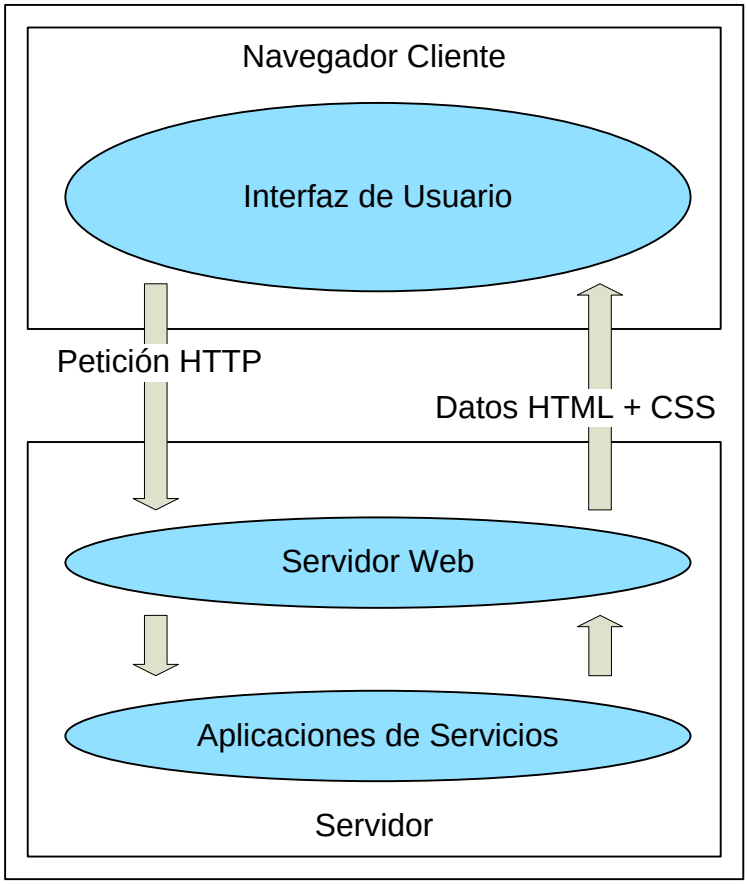


Figura 5: Flujo de navegación en las aplicaciones Web Tradicionales

2.1.4.2 Limitaciones

Entre las limitaciones de las aplicaciones Web tradicionales podemos mencionar las siguientes:

Interrupción de la operatoria del usuario y pérdida del contexto operacional: el usuario para realizar una acción con la lógica del negocio del sistema, debe cumplir el ciclo de Click – Esperar – Refrescar, esta interrupción en la operatoria lleva a una pérdida del contexto operacional.

No hay una respuesta inmediata al usuario: el tiempo de la respuesta para una petición al servidor está sujeto a la duración de las acciones desencadenadas, en consecuencia el usuario espera un tiempo sin saber el estado del sistema o el avance del proceso.

Restringidas por las posibilidades del lenguaje HTML: las aplicaciones Web tradicionales están construidas en lenguaje HTML puro, con algunas alternativas de scripting.

2.2 RIA

2.2.1 Definición

Las aplicaciones RIA (Rich Internet Applications) forman un nuevo tipo de experiencia del usuario de Internet que es más interactiva, de mejor respuesta y más atractivas que las iteraciones con aplicaciones Web HTML tradicionales.
RIA combina las ventajas de la interactividad directa y respuesta inmediata común a las tradicionales aplicaciones de escritorio con la amplia distribución de las aplicaciones Web [11].
Las aplicaciones RIA típicamente transfieren el procesamiento necesario para la interface de usuario a la capa del cliente Web pero mantienen la mayor parte de los datos (estado del programa, datos, etc.) en la capa del servidor de servicios.
La figura 6 muestra el procesamiento de datos en una aplicación RIA, si bien existe comunicación entre el cliente y el servidor para realizar el procesamiento, esta interacción es menor que en las aplicaciones Web tradicionales (ver figura 2) y se agrega la comunicación entre el navegador y el "motor cliente" RIA.

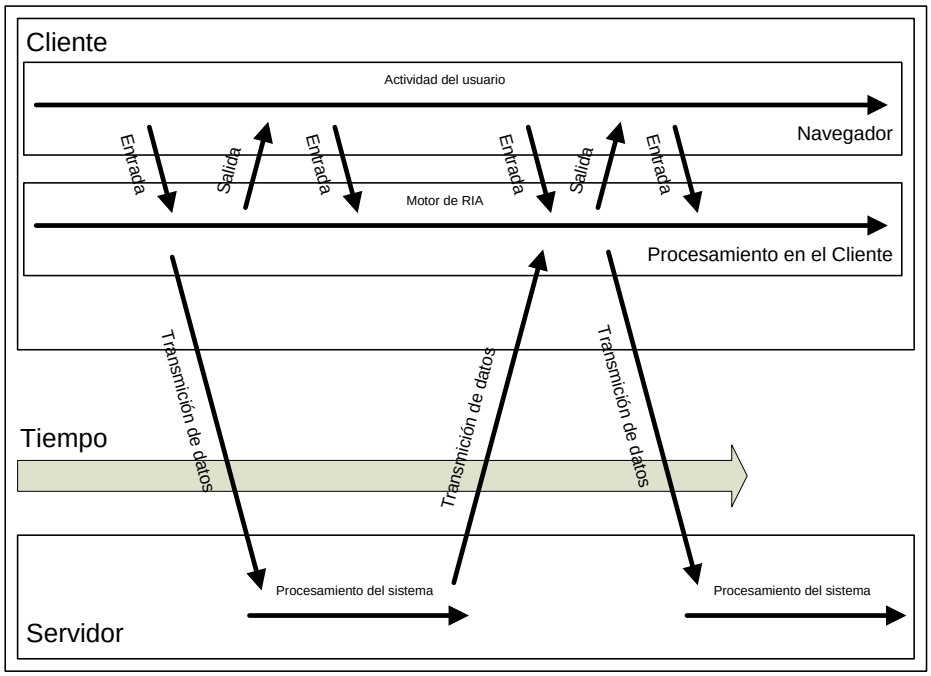


Figura 6: Procesamiento de datos en aplicaciones RIA.

2.2.2 Ventajas

Las aplicaciones Web tradicionales están construidas sobre un modelo basado en páginas de HTML. Un servidor envía contenido al navegador del cliente, página por página.

Cada vez que una actualización es necesaria, una nueva página entera es enviada por el servidor, y el usuario espera que la nueva página sea cargada.

Este modelo funcionó bien cuando los sitios Web solo mostraban textos e imagines, pero pronto la gente quiso tomar beneficios de la naturaleza distribuida de la Web, y nacieron las aplicaciones Web.

Las aplicaciones Web trajeron mayor funcionalidad a la Web, pero continuaban limitadas por el modelo de páginas y el conjunto de controles limitados e interacciones que trabajaban en estas limitaciones.

Las aplicaciones RIA liberaron a la Web del modelo de páginas, permitiendo actualizar partes específicas de la página en lugar de una recarga total de la misma. Información adicional, mensajes de error y mensajes de confirmación

pueden aparecer en la página o en capas adicionales para proveer una respuesta en lugar de abrir otra página o una ventana tipo pop-up [11].

Esta habilidad para actualizar parte de una página permite la creación de un conjunto más rico de controles, dejando que los usuarios interactúen con los elementos de la página a través de Drag & Drop, editar texto directamente, cambiar el tamaños de los objetos, y otras técnicas.

Los cambios parecen suceder inmediatamente porque las aplicaciones RIA pueden cargar datos del servidor en un segundo plano antes de que el usuario los solicite. Por ejemplo, la mayoría de los sitios de mapas cargan mapas cerca de la zona que está siendo visualizada.

Esto permite a los usuarios hacer Click y Drag al mapa para inmediatamente ver otras partes del mapa sin demora, brindando un sentido de interacción directa y respuestas inmediatas.

Este tipo de interacciones naturales y respuestas inmediatas hacen a las aplicaciones RIA tan atractivas.

2.2.3 Ejemplos

La funcionalidad “rica” en Internet puede ser usada para crear una aplicación entera, una sección dentro de una aplicación Web tradicional, o simplemente elementos en una página Web.

Un ejemplo de esto son los elementos interactivos que Yahoo agrego al diseño de su página principal a mediados del año 2006.

El contenido está dividido en Tabs que muestran más enlaces y contenido sin refrescar la página, lo cual simplifica y mejora la experiencia del usuario. Moviendo el mouse sobre los iconos en la parte superior derecha de la pantalla (Mail, Clima, Mensajería, Radio, etc.) hace que el panel se expanda para mostrar al usuario los correos más recientes, una caja de texto para buscar el clima, etc.

La figura 7 muestra la antigua página de Yahoo para poder compararla visualmente con la nueva página que incluye funcionalidad RIA.

Figura 7: Vieja página de Yahoo.

2.2.3.1 La nueva página de Yahoo

En la figura 8 se muestra la nueva página, donde los Tabs muestran contenido adicional sin refrescar los paneles de la página. Los paneles se abren para mostrar la información adicional en la misma página, el pronóstico del clima, los cinco mail más recientes, etc.

Figura 8: La nueva página de Yahoo.

2.2.3.2 Riesgos potenciales en las experiencias del usuario

Aunque las aplicaciones ricas pueden mejorar mucho la experiencia del usuario, existen algunos riesgos [30].

Como con toda nueva técnica existe siempre la tentación de sobre utilizarla y abusar de ella. Las interfaces ricas deben ser usadas cuando agregan valor a la experiencia del usuario, no para agregar campanas y silbatos o mostrar que el sitio es “cool”.

Tener conocimiento sobre los usuarios y sus tareas ayudaran a definir donde es mejor utilizar funcionalidad rica.

La gente está acostumbrada a las convenciones comunes de la Web, a pesar de sus limitaciones.

Llevará tiempo para la mayoría de las personas acostumbrarse a nuevos controles y nuevos paradigmas de interacción.

Mientras tanto es importante asegurar que los nuevos controles e interacciones son intuitivas y usables.

Como las aplicaciones RIA son bastante nuevas, la gente no siempre las reconoce como diferentes a las aplicaciones Web tradicionales, y tratarán de usar el botón de ir hacia atrás y las páginas marcadas.

Sin el concepto de páginas individuales, el botón de ir hacia atrás y las páginas marcadas naturalmente no funcionan en RIA.

Por suerte, existen formas de solucionar estos problemas.

Actualizando parte de la página puede ser mucho menos obvio y evidente que recargar la página completamente, así que es muy importante asegurarse que la gente descubra estos cambios en las páginas.

Capítulo 3. Tecnologías RIA

3.1 XMLHTTPRequest

XMLHTTPRequest es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores Web. Para los datos transferidos se usa cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON, HTML y codificaciones particulares específicas.

La interfaz se presenta como una clase de la que una aplicación cliente puede generar tantas instancias como necesite para manejar el diálogo con el servidor.

Es posible realizar peticiones sincrónicas y asíncronas al servidor; en una llamada asíncrona el flujo de proceso no se detiene a esperar la respuesta como se haría en una llamada sincrónica, si no que se define una función que se ejecutará cuando se complete la petición: *un manejador de evento*.

Internet Explorer en Windows, Safari en Mac OS-X, Mozilla en todas las plataformas, Konqueror en KDE, IceBrowser en Java, y Opera en todas las plataformas incluyendo Symbian proveen un método para el javascript del lado del cliente para realizar una petición HTTP.

El objeto hace muchas cosas más fáciles y ordenadas que lo que otros podrían hacer, e introduce otras que cosas que de otra manera serían imposibles, como por ejemplo las peticiones de cabecera para conocer cuando un recurso fue modificado por última vez, o saber si aún existe.

Hace que las opciones de los scripts sean más flexibles permitiendo que las peticiones de tipo POST no tengan que cambiar la página actual, y abre la posibilidad de usar HTTP PUT, DELETE, etc.

Estos métodos son usados cada vez más para proveer aplicaciones Web ricas como g-mail, que usa un bajo ancho de banda y ofrece una Interacción con el usuario más rápida y elegante.

En Internet Explorer, Ud. crea el objeto usando new ActiveXObject (“Msxml2.XMLHTTP”) o new ActiveXObject(“Micrisoft.XMLHTTP”) dependiendo de la versión de MSXML instalada. En Mozilla y Safari Ud. usa new XMLHttpRequest(). IceBrowser usa otro método window.createRequest().

Esto significa que se necesita mostrar diferentes *scripts* en los distintos navegadores.

El World Wide Web Consortium presentó el 27 de septiembre de 2006 el primer borrador para una especificación estándar de la interfaz. El proceso de estandarización se encuentra en su última fase desde el 27 de febrero de 2007 con la publicación del borrador definitivo.

Mientras no se alcance una versión definitiva, los desarrolladores de aplicaciones WEB pueden utilizar paquetes o framework que oculten las diferencias entre implementaciones.

La figura 9 muestra la comunicación cliente servidor utilizando la interfaz XMLHTTPRequest.

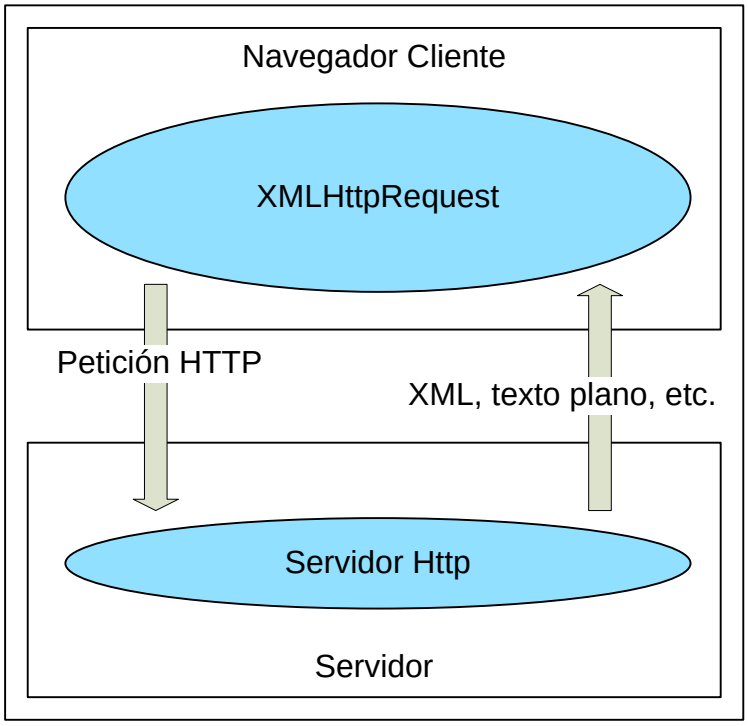


Figura 9: Comunicación mediante la clase XMLHTTPRequest.

3.2 Ajax

Ajax son las siglas de Asynchronous javascript And XML. No es un lenguaje de programación sino un conjunto de tecnologías que permiten realizar aplicaciones Web ricas [4], [6].

La característica fundamental de AJAX es permitir actualizar parte de una página con información que se encuentra en el servidor sin tener que refrescar completamente la página. De modo similar podemos enviar información al servidor.
La complejidad se encuentra en que debemos dominar varias tecnologías:

- HTML o XHTML
- CSS
- javascript
- DHTML – Básicamente debemos dominar todos los objetos que proporciona DOM.
- XML – Para el envío y recepción de los datos entre el cliente y el servidor.
- PHP, Java , ASP o algún otro lenguaje que se ejecute en el servidor.

Así la presentación es basada en estándares usando XHTML y CSS, la exhibición e interacción dinámicas usando el Document Object Model (DOM), el intercambio y manipulación de datos usando XML and XSLT, la recuperación de datos asincrónica usando XMLHttpRequest y javascript poniendo todo junto [41].
Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de un llamado javascript al motor AJAX en vez de ese requerimiento. Cualquier respuesta a una acción del usuario que no requiera un viaje de vuelta al servidor (como una simple validación de datos, edición de datos en memoria, incluso algo de navegación) es manejada por su cuenta. Si el motor necesita algo del servidor para responder (sea enviando datos para procesar, cargar código adicional, o recuperando nuevos datos) hace esos pedidos asincrónicamente, usualmente usando XML, sin frenar la interacción del usuario con la aplicación.

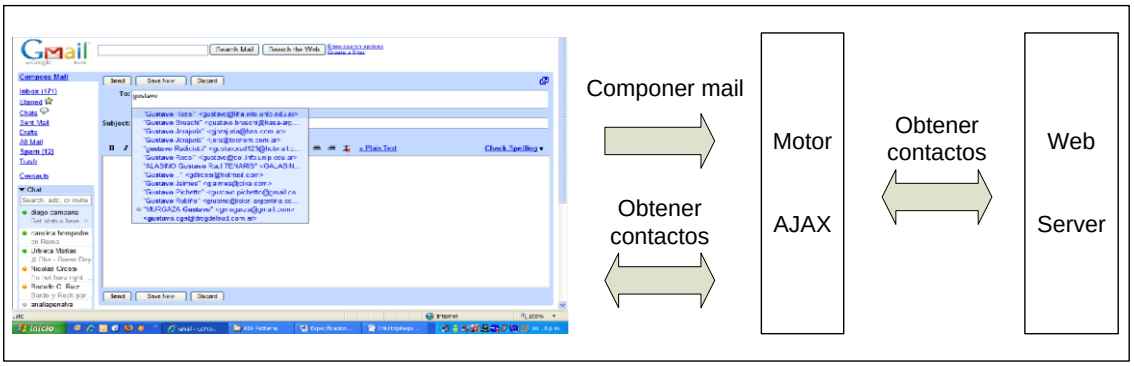


Figura 10: Interacción mediante AJAX.

La figura 10 muestra la interacción realizada con la capa del motor AJAX cuando un usuario de *gmail* busca direcciones de mail de la lista de contactos. Cuando el usuario selecciona la opción de escribir un nuevo mail, una petición asincrónica dispara la búsqueda de contactos del servidor. Este ejemplo es una de varias posibilidades de consulta asincrónica para este escenario.

3.3 Flex

Esencialmente Flex es un *framework* de Adobe para desarrollo de interfaces de usuario que agrega mucho valor para las aplicaciones RIA [10].

Aunque ActionScript 3.0 (AS3) es el lenguaje para programar aplicaciones usando el Flex framework, AS3 no es Flex. Si el código AS3 que se programe utiliza los componentes y métodos definidos en el Flex framework, entonces se está usando Flex. Es decir, AS3 no es por sí solo Flex.

Flex como resultado crea archivos SWF que son ejecutados en el motor de Adobe Flash.

Flex fue construido para realizar comportamiento rico en el cliente de la aplicación que corre en Internet y habla con servidores remotos. No fue construido para realizar aplicaciones Web o lógica en el servidor.

La figura 11 muestra un ejemplo de aplicación desarrollada con Flex es Picnik (www.picnik.com), que posee una interfaz gráfica flexible para el manejo visual de imágenes.

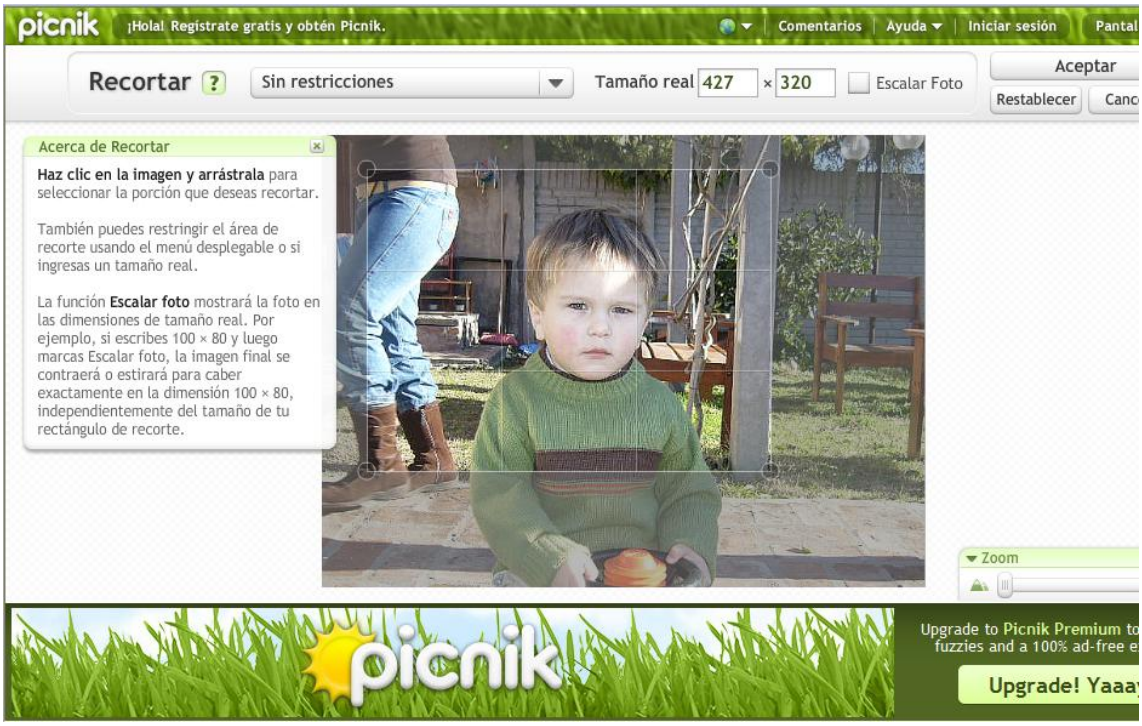


Figura 11: Sitio Web "Picnik" para edición de imágenes realizado con FLEX.

3.4 OpenLaszlo

OpenLaszlo es una plataforma de código abierto (open source) para la creación de Aplicaciones Web Ricas (RIA).

Los programas OpenLaszlo son escritos en XML y javascript, compilados transparentemente a Flash y con la versión OpenLaszlo 4, a DHTML.

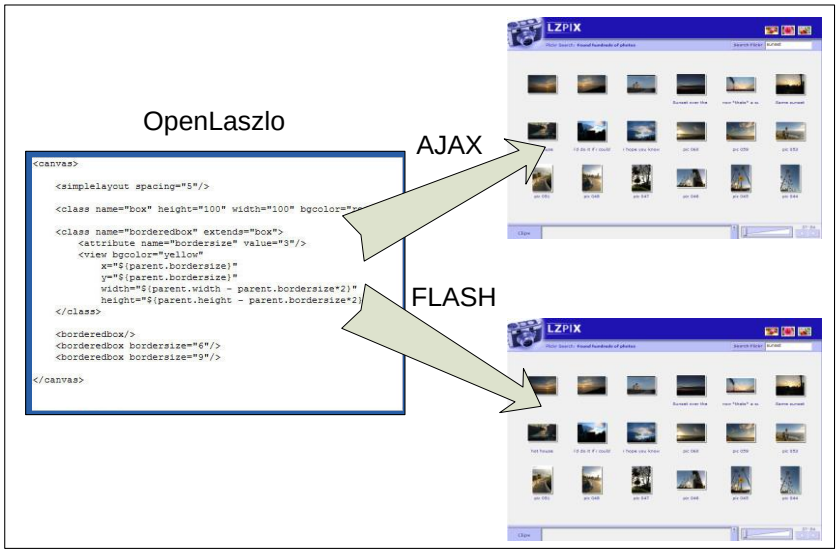


Figura 12: Generación de aplicaciones AJAX o Flash mediante la plataforma OpenLazlo.

La figura 12 representa visualmente la posibilidad de obtener una aplicación AJAX o FLASH a partir de un desarrollo utilizando OpenLaszlo.

Las APIs de OpenLaszlo proveen animación, ordenamiento visual, enlace con datos, comunicación con el servidor, e interfaz de usuario declarativa. Una aplicación OpenLaszlo puede ser tan pequeña como un único archive fuente, o factorizada en múltiples archivos que definen clases reutilizables y librerías.

OpenLaszlo es "escríbalo una vez, córralo en cualquier lugar". Una aplicación OpenLaszlo desarrollada en una máquina se ejecutará en todos los navegadores, en todos los sistemas operativos de punta del mercado [3].

Como un ejemplo de aplicación construida utilizando *OpenLaszlo* la figura 13 muestra a Pandora, una aplicación para escuchar música personalizada a través de Internet (www.pandora.com).

Figura 13: Sección del sitio web “pandora” realizada con OpenLazlo.

3.5 Comet

Es una arquitectura para aplicaciones Web en la cual un servidor Web envía datos al programa cliente (normalmente un navegador) asincrónicamente sin necesidad de que el cliente los solicite explícitamente. Esto permite la creación de aplicaciones Web basadas en eventos, permitiendo una interacción en tiempo real, que de otra manera costaría mucho más trabajo.

Las aplicaciones Comet usan conexiones HTTP de larga duración entre el cliente y el servidor, a las cuales el servidor puede responder con demora (*lazily*), enviando nuevos datos al cliente a medida que los datos están disponibles (*pushing*). Esto difiere por supuesto del modelo de las aplicaciones Web tradicionales pero también del modelo de aplicaciones Ajax, donde los navegadores solicitan paquetes de datos, utilizados para actualizar la página actual. El efecto es similar a las aplicaciones Ajax pero el tráfico de datos, la latencia y carga del servidor son menores.
Comet no se refiere a ninguna tecnología específica para lograr este modelo interactivo, pero abarca a todas ellas.

3.6 GWT

Google Web Toolkit (GWT) es un framework de desarrollo Java, que permite escapar de la matriz de tecnologías que hacen del hecho de realizar aplicaciones Ajax, una tarea difícil y propensa a errores. Con GWT se pueden desarrollar y depurar las aplicaciones Ajax en lenguaje Java, usando las herramientas de desarrollo Java a elección. Cuando se despliega la aplicación a producción, el compilador GWT traduce la aplicación Java a código javascript y HTML compatible con el navegador cliente [14].

3.6.1 El ciclo de desarrollo con GWT

- Usar el entorno de desarrollo java elegido para escribir y *debugear* la aplicación en lenguaje Java, usando tantas (o tan pocas) librerías GWT que se desean utilizar.
- Usar el compilador GWT “Java a javascript” para traducir la aplicación a un conjunto de archivos javascript y HTML que pueden utilizarse en cualquier servidor Web.
- Confirmar que la aplicación funciona correctamente en los navegadores que se requieran soportar. Esta tarea no requiere trabajo adicional, la mayoría de las veces.

3.6.2 La arquitectura GWT

GWT tiene cuatro componentes principales: un compilador “Java a javascript”, un navegador Web “hosteado”, y dos librerías de clases Java.

3.6.2.1 El compilador “Java a javascript”

El compilador “Java a javascript” traduce el lenguaje de programación Java al lenguaje de programación javascript. Se usa el compilador para correr las aplicaciones GWT en modo Web.

3.6.2.2 El navegador Web GWT “hosteado”

El navegador Web GWT “hosteado” permite correr y ejecutar las aplicaciones GWT en modo “hosteado”, donde el código corre en la máquina virtual Java sin compilarlo a javascript. Para lograr esto, el navegador GWT incrusta un control especial del navegador con un enlace a la JVM.

3.6.2.3 Librería de emulación JRE

GWT contiene implementaciones javascript de las clases java más usadas en la librería Java estándar, incluyendo la mayoría de las clases del paquete java.lang y un subconjunto del paquete java.util. El resto de las librerías de clases java no son soportados nativamente por GWT. Por ejemplo los paquetes como java.io no aplican a las aplicaciones Web ya que acceden a la red y el sistema de archivos.

3.6.2.4 La librería de interfaces de usuario GWT

La librería de interfaces de usuario GWT es un conjunto de interfaces propias y clases que nos permiten crear widgets para el navegador Web, como botones,

cajas de texto, imágenes y texto. Esta es la librería de interfaces de usuario central para crear aplicaciones GWT.

3.6.3 ¿Porque traducir código Java a javascript?

Las tecnologías Java ofrecen una plataforma de desarrollo productiva, y con GWT ellas pueden ser también una plataforma para el desarrollo de aplicaciones Ajax. Algunos de los beneficios de desarrollar con GWT:

- Se pueden usar nuestros entornos de desarrollo Java favoritos, (Eclipse, IntelliJ, JProfiler, JUnit) para desarrollar aplicaciones Ajax.
- El chequeo estático de tipos en lenguaje java impulsa la productividad y reduce los errores.
- Los errores de javascript comunes son fácilmente atrapados en tiempo de compilación en lugar de ser capturados por el usuario en tiempo de ejecución.
- Refactorización Java automatizada.
- Diseños OO basados en Java son más fáciles de comunicar y entender, haciendo el código Ajax más comprensible con menor documentación.

3.7 DOJO

Dojo es un kit *open source* de herramientas *javascript* para construir interfaces de usuario Web dinámicas. Dojo ofrece widgets, utilidades, etc. [16]

3.7.1 ¿Cuál es el punto?

Dojo se basa en *HTML* y *javascript* por lo cual el programador no tiene que usar lenguajes de programación extraños. Dojo sube el nivel de abstracción de manera que el desarrollador no tiene que reinventar la rueda cuando comienza a desarrollar un proyecto [25].

3.7.2 El sistema de paquetes

Dojo consiste de archivos *javascript.* El sistema de paquetes cuida que solo los archivos necesarios sean incluidos. Cada archivo *javascript* puede ser nombrado como un paquete y por este nombre el paquete puede ser usado. Uno no tiene que recordar el nombre de un archivo o directorio, solamente *dojo.js* debe ser incluido en el documento HTML. Este archivo tiene el código de inicialización de Dojo.

Hay un par de desarrollos pre-empaquetados que consisten en diferentes tipos de paquetes, por ejemplo: *widgets, eventos o desarrollos de interfaces*.

Aunque Dojo es agradable, bello, etc. Es un poco “pesado” y la documentación es todavía escasa.

Capítulo 4. OOHDM

4.1 Visión general

Construir aplicaciones Web complejas es difícil, a menudo porque estas aplicaciones actúan como integradores de datos distribuidos o conducen repositorios, y usualmente soportan diferentes configuraciones de usuario. Las aplicaciones Web combinan temas tecnológicos como permitir el acceso a dispositivos móviles, o soportan balanceo para versiones actuales de HTML/XML, más conceptuales como por ejemplo implementando nuevos modelos de negocio. Negocios basados en Internet pueden ser algo diferentes a los tradicionales [7].

Se debe entender el domino de la aplicación subyacente: objetos, comportamiento, reglas de negocio, etc., y construir una aplicación Web flexible y “evolucionable” en su dominio.

Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de extensión y reusabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a ésta, son de vital importancia para el desarrollo de aplicaciones complejas.

La elección de tecnologías complejas demora el proceso e incrementa los costos, pero en ocasiones permite adecuarse a metodologías de diseño más fácilmente. Tal es el caso de las tecnologías orientadas a objetos, las cuales tienden a demorar el desarrollo en etapas tempranas. El tiempo de desarrollo en la actualidad es crítico, tanto por razones de marketing como por límites en el presupuesto y los recursos, pero la adopción de estas tecnologías hace que el mantenimiento se transforme en una actividad más simple, la división en capas sea tarea natural del desarrollo y el tiempo invertido en el diseño facilite el trabajo necesario para el resto de las actividades.

Las metodologías tradicionales de ingeniería de software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia.

El tamaño, la complejidad y el número de aplicaciones crecen en forma acelerada en la actualidad, por lo cual una metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y reusabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios *Web*, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr.

En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada.

Construir la interfaz de una aplicación *Web* es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado 2. Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

OOHDM organiza el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación [27].

OOHDM propone un conjunto de tareas que en principio pueden involucrar mayores costos de diseño, pero que a mediano y largo plazo reducen notablemente los tiempos de desarrollo al tener como objetivo principal la reusabilidad de diseño, y así simplificar la evolución y el mantenimiento.

4.2 Diseño conceptual

Durante esta actividad se construye un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos.

En OOHDM, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.

Se usa notación similar a UML (Lenguaje de Modelado Unificado) y tarjetas de clases y relaciones similares a las tarjetas CRC (Clase Responsabilidad Colaboración).

El esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

Dado que el lenguaje para modelar y diagramar el modelo conceptual es UML no se ejemplifica aquí dicha etapa.

4.3 Diseño navegacional

La primera generación de aplicaciones *Web* fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia. En OOHDM, la navegación es considerada un paso crítico en el diseño aplicaciones. Un modelo navegacional es construido como una *vista* sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual.

El diseño de navegación es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son

las tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La principal estructura primitiva del espacio navegacional es la noción de **contexto navegacional**. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.

Los contextos navegacionales juegan un rol similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada.

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos las anclas), así como métodos (comportamiento) cuando se navega en un particular contexto.

4.3.1 Estructuras de acceso

Una vez que las clases navegacionales fueron determinadas, es necesario estructurar el espacio navegacional que estará disponible al usuario.

En OOHDM esta estructura es definida agrupando los objetos navegacionales en conjuntos llamados **Contextos**.

Cada definición de un contexto incluye, además de los objetos incluidos en él, la especificación de la estructura de navegación interna, un punto de acceso, restricciones de acceso en términos de las clases de usuario y operaciones, y una estructura de acceso asociada [28].

El diagrama de contexto no contiene toda la información necesaria para especificar todos los contextos: cada contexto y estructura de acceso deberá ser detallada en una tarjeta CRC.

4.3.2 Ejemplo de modelo navegacional

Para dar un ejemplo simplificado del modelo navegacional y de las siguientes capas (que son las que más interesan a este trabajo), vamos a partir de una interface existente: la sección del clima en la página de la CNN, y describir las diferentes etapas OOHDM para dicha interface.

En la misma se selecciona una zona geográfica y se obtienen diferentes vistas de la misma (pronóstico climático, imagen satelital, etc.). En este trabajo el modelo navegacional constará de una sola clase para simplificarlo.

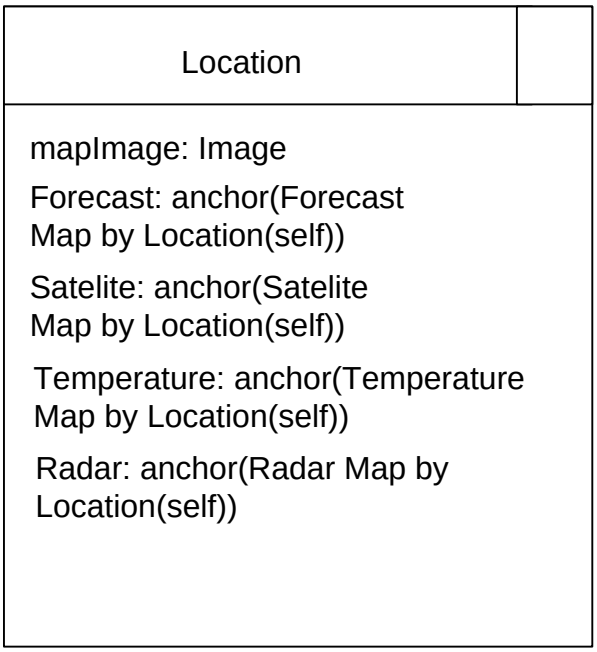


Figura 14: Modelo navegacional de la ubicación (Location).

Como se puede ver en la figura 14, la clase dispone de un atributo imagen que permite acceder a la imagen del mapa y cuatro estructuras de acceso que permiten cambiar la vista del mapa actual para la ubicación.

4.4 Interface abstracta

Una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué

transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas.

Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario.

El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones *Web*) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

4.4.1 Vistas de Datos Abstractas - ADVs

El modelo de interfaz ADV (Vista de Datos Abstracta) especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs en la pantalla real son hechas en la fase de implementación.

La interface de usuario es especificada usando ADVs, que soportan un modelo orientado a objetos para los objetos de una interface. En OOHDM se de define un ADV para cada clase de un nodo, indicando como cada atributo de un nodo o sub-nodo (si es un nodo compuesto) será percibido. Un ADV puede ser pensado como un observador del nodo, expresando sus propiedades perceptibles, en general como ADVs anidadas o tipos primitivos (por ejemplo: botones). Usando un diagrama de configuración expresamos como estas propiedades se relacionan con los atributos del nodo.

Una ADV es un objeto compuesto que posee estado y comportamiento, Las ADVs pueden ser compuestas o agrupadas en jerarquías de generalización / especialización permitiendo así algún nivel de rehúso, cuando se definen tipos de objetos recurrentes a la interface (como botones, mapas, etc.)
[34].

Son abstractas ya que son independientes a la implementación, sin embargo los detalles de los aspectos de un nivel menor (como la locación, el color de fondo, etc.) pueden anotarse en la especificación de la ADV o como parte del estado de la ADV.

Las ADV observan a los objetos de la aplicación, en los cuales la lógica de la aplicación y los datos de la aplicación son generalmente manejados. Sin embargo, siendo objetos, ellos pueden contener comportamiento arbitrario incluyendo parte de la lógica de negocio, la cual en las aplicaciones Web tradicionales se encuentra en el controlador de la arquitectura MVC. En cambio, en las aplicaciones RIA esta lógica puede ubicarse en la interface.
Un ADV puede ser el observador de un objeto de la capa de navegación o un objeto de la capa conceptual, en caso de no requerir objetos navegacionales. Un ADV puede mostrar el estado del objeto como también actuar en su comportamiento, tal es el caso de botones, listas de opciones, etc.).

4.4.1.1 Ejemplo de ADV

La figura 15 muestra el ADV correspondiente a la interface abstracta del mapa geográfico descripto en el ejemplo de capa navegacional. En dicho ADV se ubican los diferentes controles de la interface (imagen y botones) y su relación con el nodo navegacional.

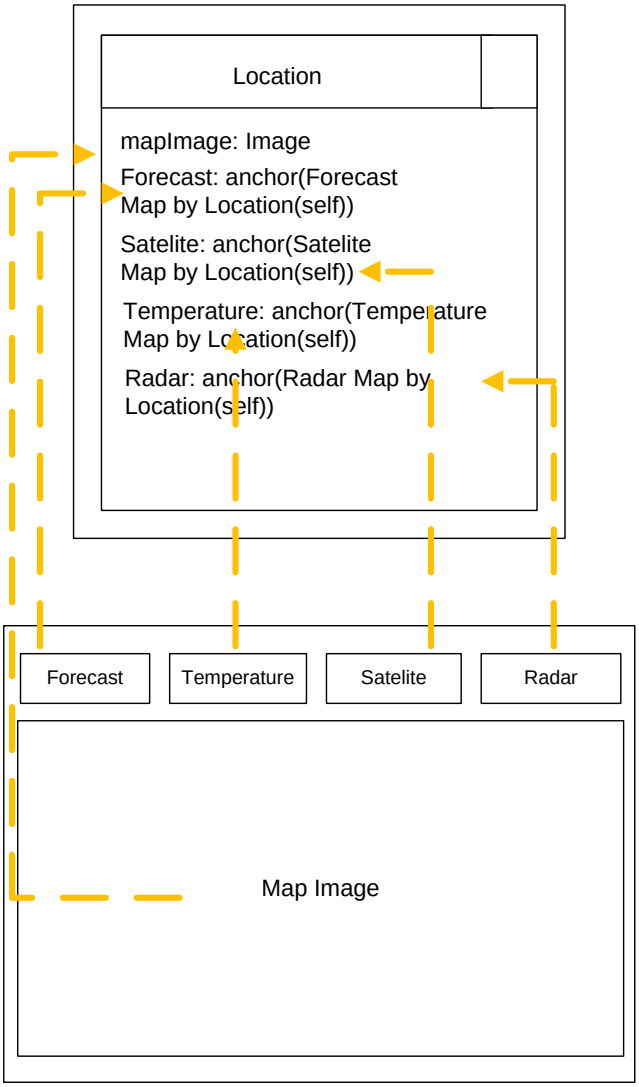


Figura 15: Modelo Interface Abstracta – ADV (mapa geográfico).

4.4.2 ADV Charts

Los ADV Charts proveen un esquema visual para la especificación de los aspectos dinámicos de una interfaz. Contienen uno o más estados y transiciones así como también pueden contener atributos y otros ADVs para describir su navegación [22].

Estos aspectos de comportamiento tales como el procedimiento interactivo y los efectos que ocurren en la interface son de mucha importancia para la especificación de aplicaciones RIA.

Los diagramas ADV son máquinas de estado que permiten expresar transformaciones de la interface que ocurren como resultado de una interacción del usuario.

Usados en conjunto con diagramas de configuración indican el camino por el cual eventos en la interface disparan operaciones en la capa navegacional o conceptual.

La nidación de estados en ADVs sigue la misma semántica que la de los diagramas de estado, mientras que la nidación de ADVs dentro de estados indican cuales ADVs se encuentran en dicho estado.

Los ADV-Charts se basan en las máquinas de estado y son una extensión de StateCharts y ObjectCharts, utilizando también notaciones de Redes de Petri.

4.4.2.1 Statecharts y Objectcharts

Statecharts y Objectcharts son formalismos visuales para describir el comportamiento de sistemas complejos. Los diagramas Statecharts son una extensión de las máquinas de estado finito, y fueron formulados para evitar muchas de las desventajas que tienen las representaciones de las máquinas de estado finito, como ser la explosión de estados y falta de estructuras [15].

Las máquinas de estado finito se describen por un alfabeto de entrada, un alfabeto de salida, un conjunto de estados (incluyendo el inicio y final) y un conjunto de transiciones. Los diagramas Statecharts extienden la notación de las máquinas de estado finito agregando jerarquía, concurrencia y comunicación. La jerarquía permite a las transiciones comunes ser agrupadas y agrega estructura al diagrama. La jerarquía es mostrada en la figura 16.a donde el estado A esta compuesto por dos estados (B y C). El sistema puede estar en el estado B o C pero no en ambos estados al mismo tiempo.

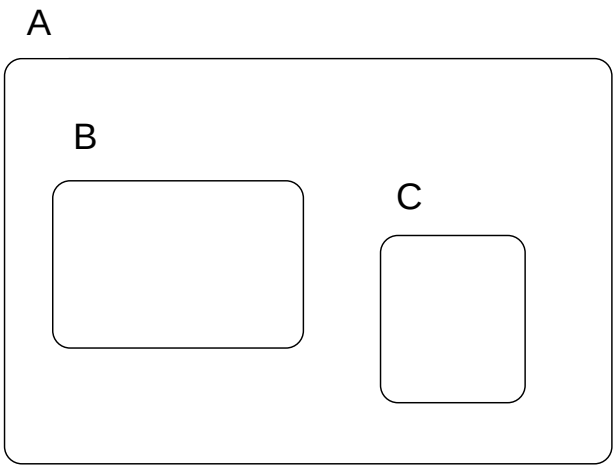


Figura 16.a: Jerarquía en Statecharts

La concurrencia es representada por una descomposición lógica (Y) de estados, y es un intento para evitar la explosión de estados. Por ejemplo, en la figura 16.b el estado D consiste de componentes E y F.

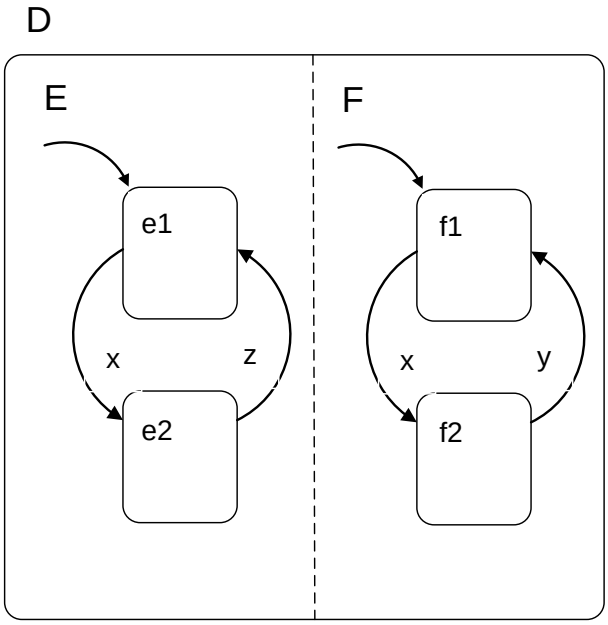


Figura 16.b: Concurrencia en Statecharts

El sistema puede estar en los estados E y F simultáneamente. Las transiciones en los estados E y F pueden ser lanzadas sincrónicamente, o independiente una de otra. En caso de que el sistema se encuentre en el estado <e1,f1> y la transición X se dispare, el sistema pasa al estado <e2,f2>. Sin embargo, si el

sistema se encuentra en el estado <e2,f2> y un evento Y ocurre, el sistema pasa al estado <e2, f1>, lo que significa que no se disparo una transición del componente E.

El mecanismo de comunicación utilizado en los diagramas Statecharts es llamado comunicación de difusión (*broadcast communication*)

La comunicación de difusión es la habilidad de propagar la ocurrencia de un evento a través de todos los componentes Y, y disparar simultáneamente todas las transiciones causadas por un evento.

La notación de Objectcharts, la cual es un formalismo visual para la especificación de sistemas orientados a objetos, usa dos tipos de diagramas. Las instancias de objetos y su comunicación son capturados por lo que significa un diagrama de configuración; el comportamiento de las clases es descripto en los diagramas Objectcharts.

Un diagrama de configuración para la especificación ADV es mostrada en la figura 17 y uno para Objectcharts es similar. Los servicios provistos y requeridos por una clase son mostrados en el diagrama.

Los Objectcharts caracterizan el comportamiento de una clase como una máquina de estado, con un alfabeto de entrada y salida comprendiendo las operaciones provistas y requeridas por la clase. Para diferenciar los servicios provistos de los requeridos, el prefijo “/” es usado en todo lugar que un servicio provisto sea referenciado en una transición. Los Objectcharts no usan comunicación de difusión, en su lugar, consideran la petición de un servicio y la ejecución completa de un servicio como un evento atómico único.

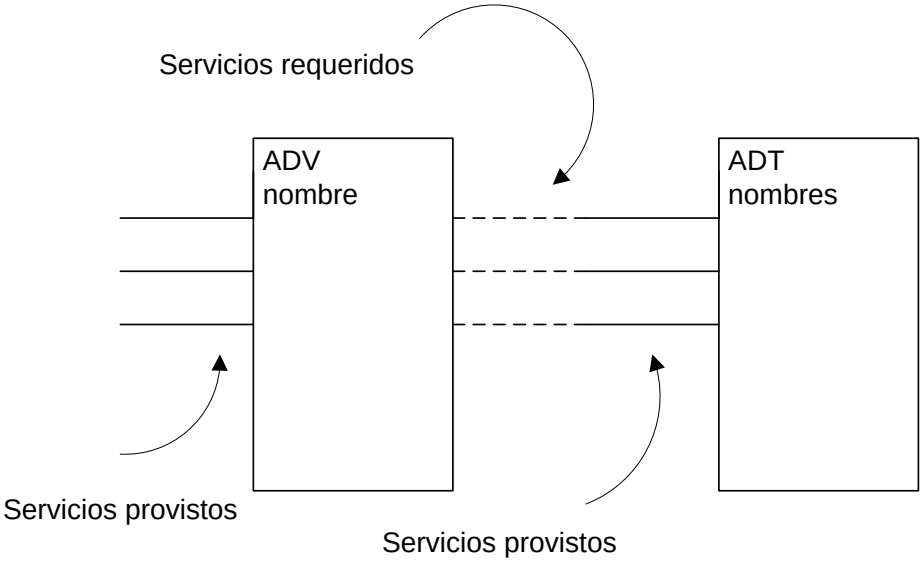


Figura 17: Diagramas de configuración para un ADV

Los diagramas Objectcharts aumentan los estados con atributos (variables y sus correspondientes valores) y *observadores*. El efecto de las transiciones de estado en los atributos son especificadas. Los observadores permiten a un servicio provisto informar sobre los valores de atributos sin cambiar el estado de un objeto.

Para completar un Objectchart, las transiciones deben especificarse. La especificación de una transición comprende los nombres de los estados inicial y final de la transición, el nombre del servicio para la transición, la condición que dispara la transición y una post-condición.

La condición que dispara la transición es un predicado que describe las restricciones sobre atributos y observadores para la transición. También podemos agregar a los Objectcharts especificaciones invariantes, comprendiendo el nombre del estado y la relación que necesita cumplir en el estado que estamos describiendo.

Cada máquina de estado finita tiene un estado de comienzo que es su estado por defecto. Este estado se diferencia de otros estados, en Statecharts y Objectcharts, anotándolo con una flecha pequeña sin un estado inicial. Usamos la misma representación en los diagramas ADV-Charts.

4.4.3 ADV y ADV-Charts

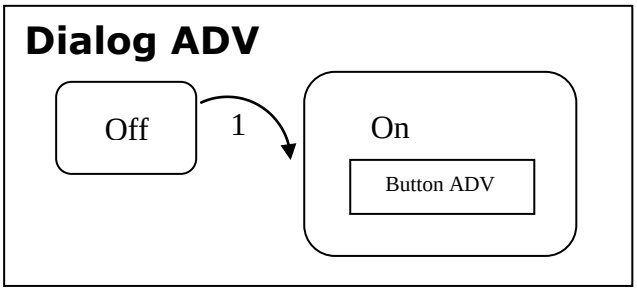
Los diagramas ADV-Charts son parte de las especificaciones ADV [19].
Proveen un esquema visual para especificar sistemas interactivos que se basan en el concepto de ADV.
Como se mencionó anteriormente, los alfabetos de entrada y salida para los Objectcharts son el conjunto de servicios provistos y requeridos. Como los ADV-Charts se basan en máquinas de estado finitas, es necesario definir el alfabeto de entrada y salida.
Los eventos externos (iniciados por el usuario) son modelados como servicios provistos por el ADV, porque el ADV provee servicios al usuario que genera el evento. Es notable la necesidad de asociar las transiciones con el estado del sistema en lugar de asociarlas únicamente con los servicios provistos y requeridos. De tal modo, se asocian condiciones sobre atributos (variables y sus valores) con transiciones. De esta manera, se puede indicar que una transición se dispara porque el estado del ADV cambia, sin tener que incluir un evento artificial para este propósito.
En consecuencia, el alfabeto de entrada a un ADV-Chart consiste de los servicios provistos y requeridos por el ADV, y las condiciones sobre atributos del ADV. El alfabeto de salida de un ADV-Chart consiste en los servicios provistos y requeridos por un ADV y otras funciones indicadas dentro del ADV.
El concepto de ADV soporta anidamiento de comportamiento y estructural. Anidación de comportamiento es representada en ADV-Charts a través del anidamiento de estados, y es también soportado por Statecharts y Objectcharts. El anidamiento estructural es representado en ADVCharts a través del anidamiento de ADVs. Esto no es representado en Objectcharts o Statecharts.
Los ADVCharts generalizan a los Objectcharts y Statecharts para manejar aspectos específicos de diseño a sistemas interactivos, como usar dispositivos de punteros para asociar eventos a ADVs particulares, y la concurrencia inherente entre los ADVs que componen la interface de usuario. El anidamiento estructural soporta indicar el foco de control como parte del diseño ya que los componentes de un objeto están claramente indicados. Dada la concurrencia

inherente de los ADVs, un mecanismo para sincronización de ADVs es necesario, ya que estamos especificando las interfaces de usuario como una colección independiente de ADVs. Con este propósito se ha introducido algunas notaciones de redes de Petri en los ADV-Charts con el fin de representar la sincronización.

4.4.3.1 Símbolos y reglas usados en los ADV-Charts

Un ADV es representado en los ADV-Charts por un rectángulo con el nombre del ADV en la parte superior del rectángulo.
Un estado es representado por un rectángulo de esquinas redondeadas conteniendo el nombre del estado. Los ADVs pueden contener uno o más ADV-Charts para describir su comportamiento y pueden contener otros ADVs para describir su descomposición por estructura. Un estado puede estar solo, puede contener un grupo de estados, de ADVs, o ambos.
Los atributos pueden definirse en el ADV o en un estado. El alcance de un atributo definido en el ADV es el ADV mismo; es decir, todos los estados y ADVs definidos dentro de este ADV ven el atributo. El alcance de un atributo definido en un estado; es el estado en sí, es decir, todos los estados definidos dentro de él, incluyendo estados internos a los ADVs definidos dentro de él.
Como en los StateCharts los diferentes estados son unidos por transiciones. Las transiciones son flechas de un estado a otro.
Las transiciones de un ADV a un estado, de un estado a un ADV, o de un ADV a otro, no están permitidas.
El lenguaje usado para definir transiciones es VDM.
Una transición en un ADV-Chart es etiquetada con su nombre.

- El evento al cual está respondiendo.
- La pre-condición que es requerida.
- La post-condición que genera (generalmente otro evento que disparará un cambio de estado o una acción).



1:
Event:
Pre-Cond:
Post-Cond:

Figura 18. Ejemplo de especificación mediante ADV Charts.

La figura 18 muestra un ejemplo simple de transición de un estado “Off” a un estado “On”.

Para disparar una transición las condiciones que deben cumplirse están dadas en la pre-condición.

El campo evento especifica el evento que disparará la transición (por ej. Un Click del mouse). La post-condición da las condiciones que se mantienen una vez que la transición fue disparada.

Tanto la pre-condición como la post-condición son expresiones booleanas.

Las expresiones booleanas pueden ser formadas por métodos de objetos y operaciones booleanas: y “&”, o “|”, y la negación.

Existen funciones acordadas para indicar determinadas acciones, tal es el caso de **focus** que indica la posición del cursor, y la seudo-variable **perCont** (en referencia al contexto percibido) para indicar los objetos que son agregados o quitados del contexto [34].

Los ADVs también poseen variables de estado que indican su posición por defecto; esta posición puede ser además indicada como parámetros de las operaciones al contexto (perCont) cuando se especifica la post-condición.

En el siguiente capítulo se ampliará con ejemplos el uso de símbolos (y su semántica) en ADV-Charts.

4.4.3.2 Ejemplo de ADV Chart

Para mostrar un ejemplo simple de ADV Chart, se usará el mismo ejemplo anterior. Se mostrará el comportamiento del componente mapa en el cliente, sin necesidad de refrescar la página ante un cambio de vista.
Para ir paso a paso se muestran (figura 19 a) primero los dos estados primarios (que algunas veces son obvios). En este ejemplo se especifican: off/on (apagado/encendido).
El componente inicialmente se encuentra en el estado **off**.

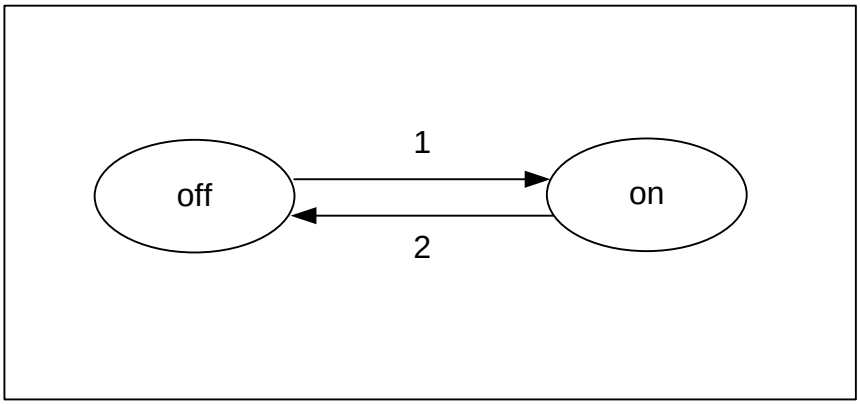


Figura 19 (a): Modelo Interface Abstracta – ADV Chart (componente de mapa gráfico por localidad con distintas vistas).

Ante el evento (1) “mouseClick” sobre el botón GO (no mostrado en el ALV por simplicidad) se inicia la búsqueda de información meteorológica de la localidad ingresada en un cuadro de texto; como resultado se despliega el mapa geográfico con la vista inicial FORECAST.

1:
Event: MouseClick
Pre-cond: Focus(goButton)
Post-cond: perCont = perCont + locationMap.showForecast(location)

El componente pasa al estado **off** mediante la ejecución del segundo evento (2) que es también “mouseClick” pero cuando se realiza sobre un “link de salida” (cualquiera sea este). El resultado es la sustracción del mapa gráfico del contexto.

2:

Event: MouseClick

Pre-cond: Focus(quitLink)

Post-cond: perCont = perCont - locationMap

Dentro del estado **on**, el componente incluye otros estados, es decir que puede estar en alguno de ellos en algún momento determinado. El estado inicial es FORECAST, desde el cual puede pasar a cualquiera de los estados TEMPERATURE, SATELITE, RADAR, mediante eventos de selección de cambio de vista. Se puede ver parte del nuevo diagrama ADV-Chart y uno de los eventos como ejemplo en la figura 19 (b).

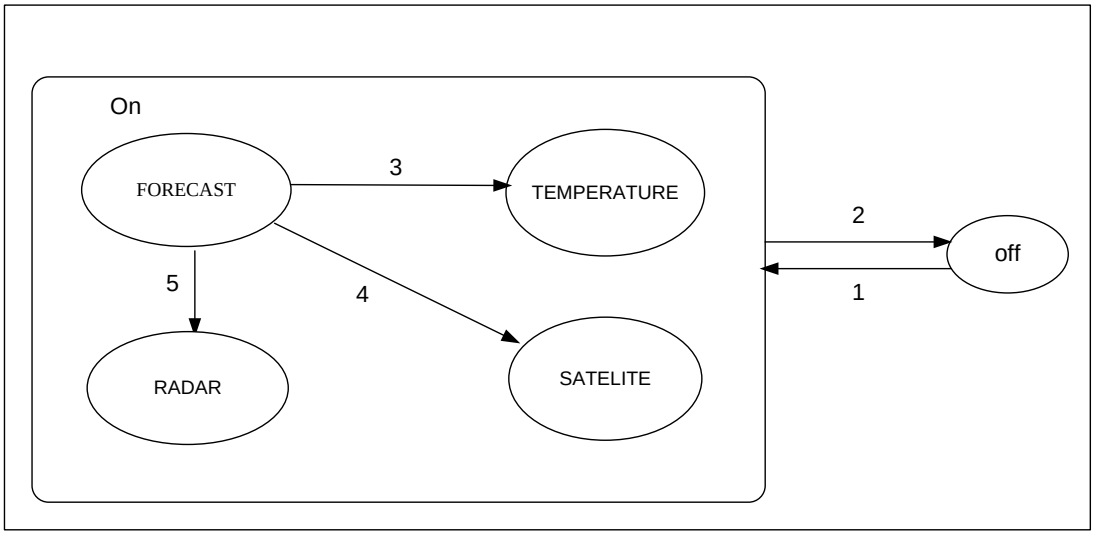


Figura 19 (b): Modelo Interface Abstracta – ADV Chart (componente de mapa gráfico por localidad con distintas vistas).

Al hacer click con el mouse sobre el Tab correspondiente a la temperatura el contenido del mapa cambia y muestra la vista por temperatura para la localidad seleccionada en el inicio.

3:

Event: MouseClick

Pre-cond: Focus(TemperatureTab)

Post-cond: perCont = perCont + locationMap.showTemperature(location)

De igual forma es posible volver al estado FORECAST o navegar en todas las direcciones posibles entre los 4 estados. El diagrama completo ADV-Chart se puede ver en la figura 19 (c).

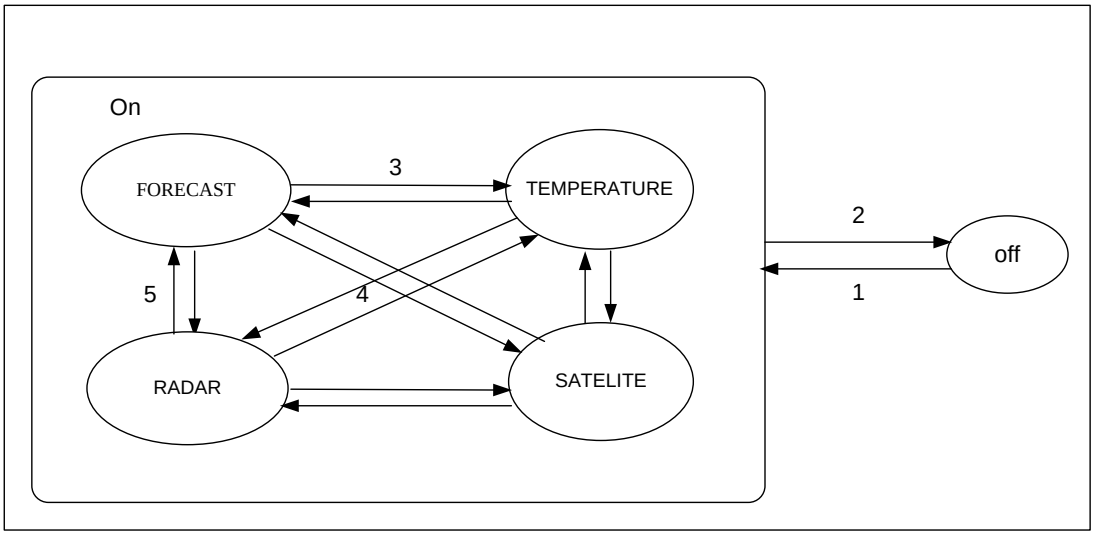


Figura 19 (c): Modelo Interface Abstracta – ADV Chart (componente de mapa gráfico por localidad con distintas vistas).

4.5 Implementación

En esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse. A fin de implementar todo en un entorno *Web*, el diseñador debe decidir además qué información debe ser almacenada.

4.5.1 Ejemplo de implementación

En la figura 20 (a) se muestra la implementación del componente de visualización para diferentes vistas del mapa geográfico de una ubicación y la

relación con su ADV como fue descripta en los ejemplos anteriores de este capítulo.

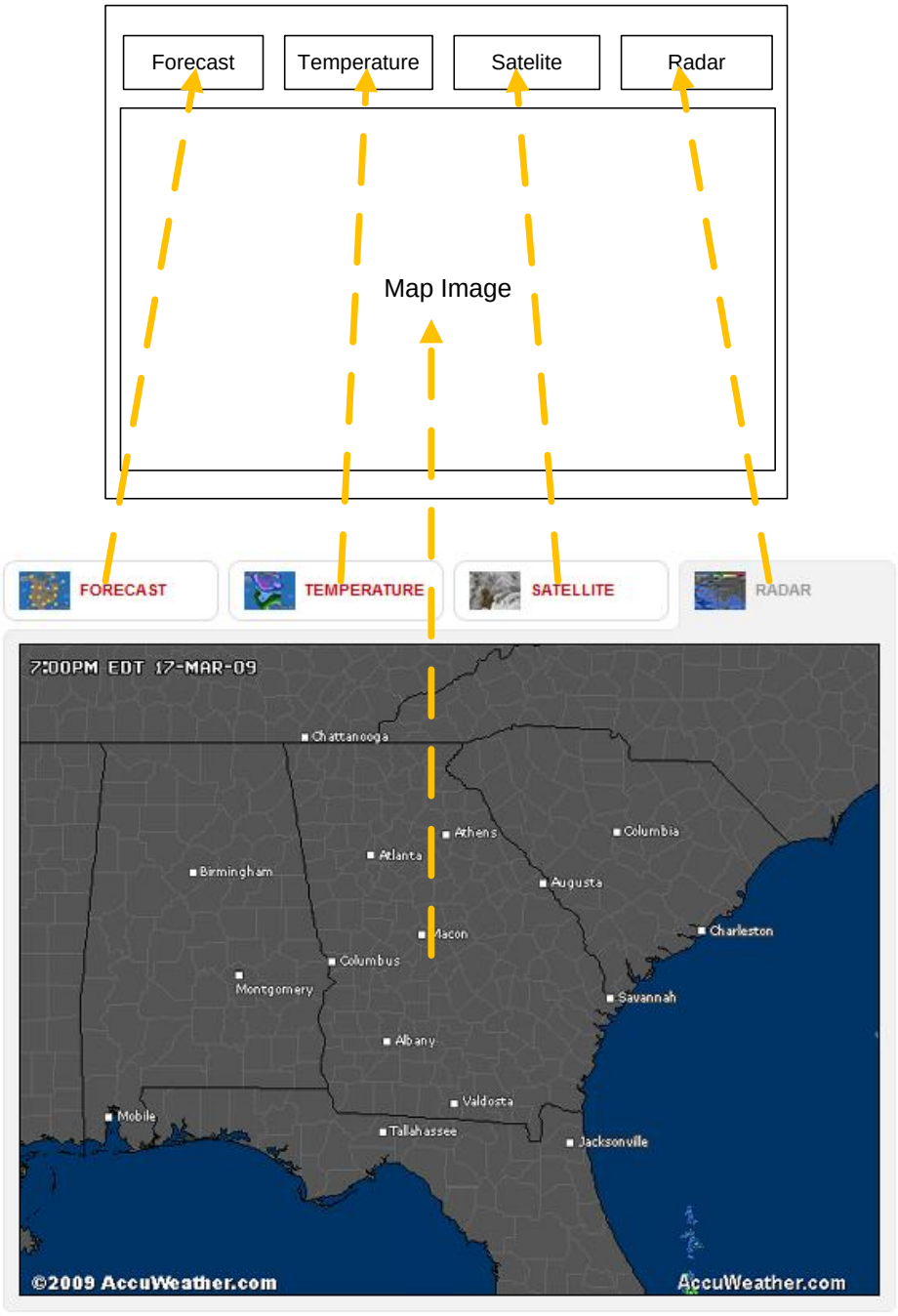


Figura 20 (a): Implementación – Diferentes vistas de mapas de una ubicación: http://weather.edition.cnn.com/weather/intl/forecast.jsp?iref=wxglobaldefault

En esta implementación del componente existe otro comportamiento que está presente en la interface, es la ayuda desplegada (hint) sobre los botones cuando se detiene el mouse sobre ellos (figura 20 b).

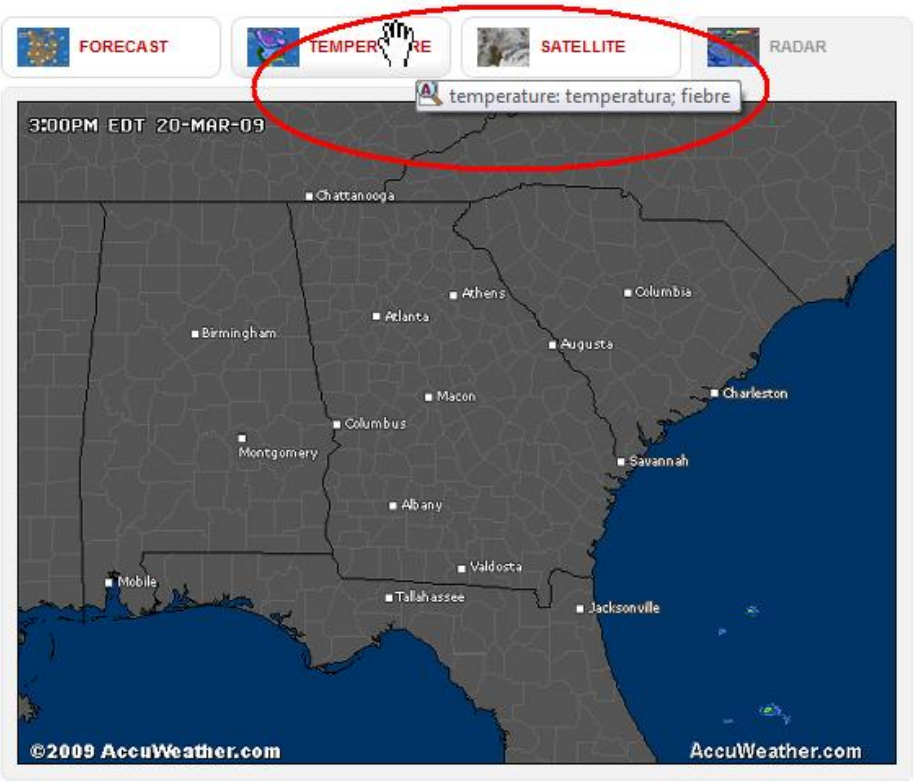


Figura 20 (b): Implementación – Cuadro informativo desplegado sobre los botones.

El correspondiente ADV-Chart incluyendo este comportamiento se muestra en la figura 21.

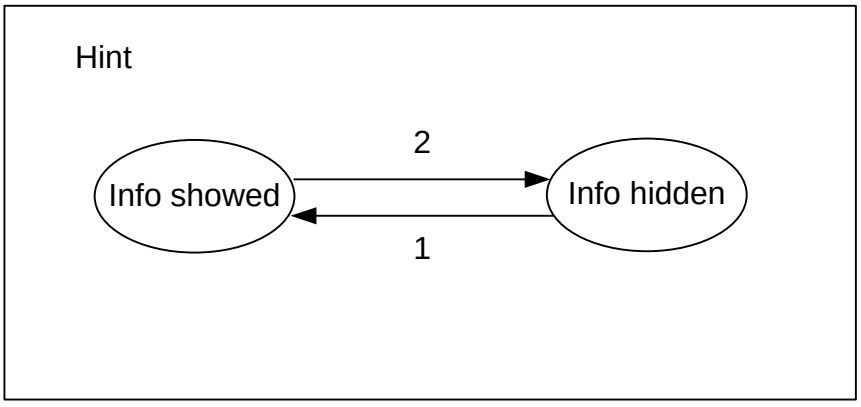


Figura 21: ADV-Chart comportamiento de información desplegable sobre los botones (hint).

El evento 1: cuando el foco se encuentra en uno de los botones y se dispara el evento MouseOver, al contenido perceptible se agrega el “hint” correspondiente a dicho botón.

1:
Event: MouseOver
Pre-cond: Focus(Button(i))
Post-cond: perCont = perCont + button(i).hint()

El evento 2: cuando el foco se encuentra en uno de los botones y se dispara el evento MouseOut, al contenido perceptible se sustrae el “hint” correspondiente a dicho botón.

2:
Event: MouseOut
Pre-cond: Focus(Button(i))
Post-cond: perCont = perCont - button(i).hint()

Capítulo 5. Especificación de Patrones Web en aplicaciones RIA

5.1 Introducción

En Ingeniería de Software como en el resto de las ingenierías el trabajo de especificar los requerimientos y tareas a realizar es sin duda uno de los más importantes del proceso de desarrollo.
El objetivo de este capítulo es plantear una forma de especificar los patrones Web de interfaces RIA, tomando como base las especificaciones actuales en bibliografía y sitios dedicados a documentar patrones Web, agregando a los mismos diagramas ADV y ADV-Charts para ampliar su descripción y mejorar la calidad de la misma.
Se hará un repaso de las especificaciones de patrones de diseño de aplicaciones OO comúnmente llamados patrones GAMA o GoF [8] y se verán los puntos que aplican generalmente a los patrones RIA.
En el capítulo siguiente se mostrará como especificar los cambios en la arquitectura de capas para la transformación de una aplicación Web tradicional a RIA.
Y en el capítulo 7 se verá un posible uso y aporte de ADV-charts a los test de una aplicación web RIA.

5.2 ¿Qué es un patrón de diseño?

"Un patrón de diseño en Software es una solución general y reusable a un problema que ocurre frecuentemente en el diseño de Software."

5.2.1 Tienen como objetivos

- Guardar experiencias de diseño OO
- Nombrar, evaluar y explicar diseños importantes que se repiten en los sistemas OO

- Ayudar a escoger alternativas de diseño (las que hacen los sistemas reusables y evitar las que no)
- Mejorar la documentación y el mantenimiento
-

5.2.2 ¿Cómo se describe un patrón de diseño?

La descripción de un patrón de diseño se compone de dos partes:

El contexto de uso →

- guías de uso, principios, compromisos, consecuencias, etc.

Descripción del problema y la solución →

- estructura, participantes y colaboraciones.
- sinónimos y ejemplos.

Un patrón de diseño se describe generalmente con la siguiente información:

- **Nombre y Clasificación**
- **Propósito**
  Que hace el patrón
- **Otros nombres**
- **Motivación**
  Ejemplo real y cómo las clases y objetos del patrón lo resuelven
- **Aplicaciones**
  Cómo identificar situaciones donde utilizarlo
- **Estructura**
  Diagrama de clases y traza de eventos (OMT aumentado)
- **Participantes**
  Lista de clases y objetos que participan con sus responsabilidades
- **Colaboraciones**
  Como colaboran los objetos para llevar a cabo sus responsabilidades
- **Consecuencias**
  Ventajas y desventajas de utilizar el patrón
- **Implementación**

Código de ejemplo

- **Usos conocidos**
- **Patrones relacionados**

5.2.3 Categorías

Los patrones de diseño comúnmente se clasifican dentro de las siguientes categorías de acuerdo a su propósito y sentido.
Patrones de **creación**: abstraen el proceso de creación de objetos. Flexibilidad en qué se crea, quién lo crea, como se crea y cuando se crea.

- Factoría Abstracta, Builder, Método Factoría, Prototipo, Singleton, …

Patrones **estructurales**: clases y objetos que se combinan para formar estructuras más complejas.

- Patrones basados en herencia (Adapter, …)
- Patrones basados en composición (Composite, Decorator, …)

Patrones de **comportamiento**: relacionados con la asignación de responsabilidades entre clases.

- Patrones basados en herencia (Interpreter, Template Method, …)
- Patrones basados en composición (Mediator, Observer, …)

5.3 ¿Qué es un patrón de diseño Web?

La definición de un patrón de diseño Web es en realidad una especialización de la definición de patrón de diseño:
“Un patrón de diseño Web es una descripción de un problema de diseño Web y buenas soluciones a este problema”.

El problema que viene a solucionar el patrón es un problema de interface de usuario, de usabilidad o prestación en la misma, mejorando tanto la funcionalidad del sitio como la amigabilidad del mismo.

Problema: el usuario necesita saber su ubicación dentro de la estructura jerárquica de un sitio Web para poder navegar atrás a una jerarquía mayor en la estructura.

Solución: pestañas de navegación → Breadcrumbs (migas de pan)

La figura 22 muestra un sitio que utiliza el patrón Breadcrums como parte de su navegación.

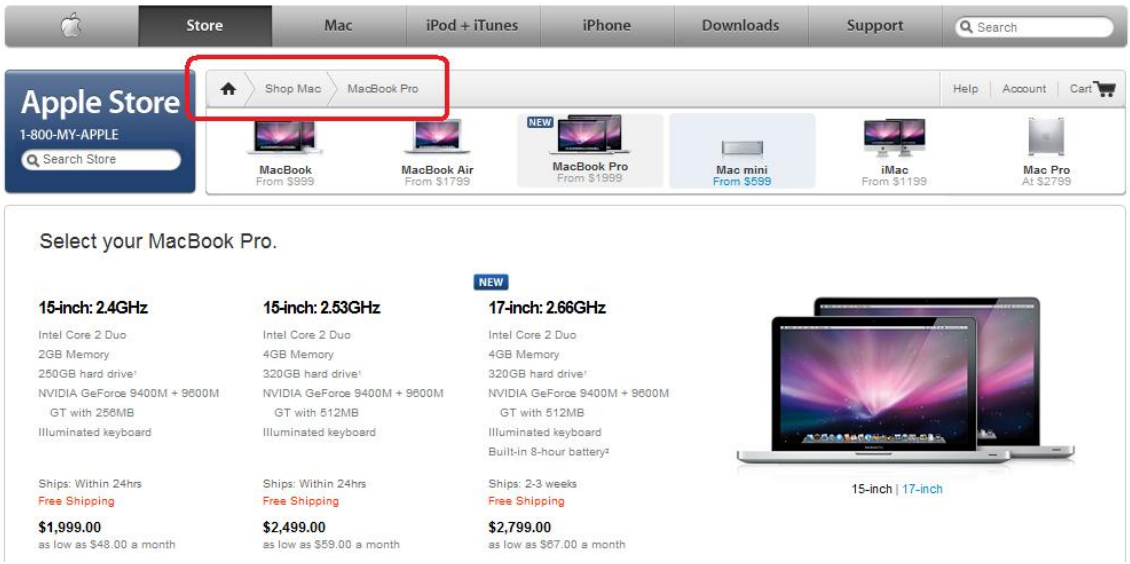


Figura 22: Patrones de diseño Web embebidos en una pantalla - Breadcrumbs.

5.3.1 Categorías

Los patrones de diseño Web en la comunidad informática son agrupados en diferentes categorías que varían según el sitio, persona o grupo que los categorice [35].

Algunas categorías que comúnmente se encuentran y coinciden son: formularios, organización de información, navegación, búsqueda, personalización, shopping o compras, etc.

Por ejemplo la librería de patrones de Yahoo [32] describe una gran cantidad de patrones Web agrupados en categorías.

5.3.2 ¿Cómo se describe un patrón de diseño Web?

Un patrón de diseño se describe generalmente con la siguiente información dentro de la bibliografía y sitios especializados. Los puntos de la descripción son similares a los utilizados para los patrones de diseño OO mencionados anteriormente.

- **Nombre y Clasificación**
- **Sinónimos**
  Otros nombres utilizados para el mismo patrón
- **Resumen del problema**
- **¿Cuándo usar?**
  Cuando se recomienda utilizar el patrón
- **Solución... ¿Cómo?**
  En qué consiste la solución propuesta por el patrón
- **Justificación... ¿Porqué?**
- **Casos especiales**
- **Patrones relacionados**
- **Ejemplos**

Todos estos puntos si bien dan una idea de la necesidad del patrón, no son suficientes para que un diseñador cuente con una especificación a partir de la cual construir un componente (que implemente el patrón). Algunas especificaciones incluyen seudocódigos o fragmentos de programas como ejemplos, imágenes mostrando diferentes estados del componente, etc.
La propuesta presentada en este trabajo es completar las mismas con una definición gráfica de la interface del patrón usando ADVs y una definición del comportamiento del patrón mediante ADV-Charts.

5.4 Inclusión de ADV y ADV-Charts en la especificación de patrones de diseño Web.

El aporte de este trabajo es mejorar las especificaciones actuales de patrones Web partiendo del siguiente punto: una de las propiedades deseables de una especificación es la no-ambigüedad, es decir que no puede existir más de una interpretación a partir de lo especificado.

Las ambigüedades se introducen fácilmente en una especificación, especialmente con el uso de lenguaje natural y haciendo descripciones informales.

Esta misma propiedad es aplicable a la especificación de patrones de diseño Web.

Para evitar ambigüedades se necesitan notaciones que puedan especificar patrones de diseño en forma sencilla y práctica. Existen diversos lenguajes formales para realizar especificaciones en Ingeniería de Software, aplicados a patrones de diseño podemos mencionar LePUS3 and Class-Z [9], Role Based Metamodeling Language (RBML) [18], State Machine Pattern Specification (SMPS).

Las especificaciones informales son más fáciles de construir pero proveen menos soporte al análisis del sistema. En este sentido, no hay certeza que las especificaciones son completas, no ambiguas, o que cumplen todas las características deseadas de los requerimientos funcionales y no funcionales.

Sin embargo, mucha gente utiliza especificaciones informales por la errónea creencia que los métodos formales son aún más costosos y difíciles.

Las especificaciones formales requieren un mayor tiempo de especificación y experiencia pero permiten y soportan el análisis de especificaciones y disminuyen los esfuerzos en las pruebas (ver…).

Las especificaciones algebraicas son una manera formal de especificar sistemas de software. Esta técnica ha sido desarrollada por tres décadas y es muy conocida.

Dentro del plano formal las especificaciones pueden además ser clasificadas como textuales y gráficas.

Las especificaciones gráficas son generalmente más claras que las especificaciones textuales pero menos escalables.

Como se puede ver en la figura 23, la característica textual o gráfica de una especificación es ortogonal a la formalidad de la especificación.

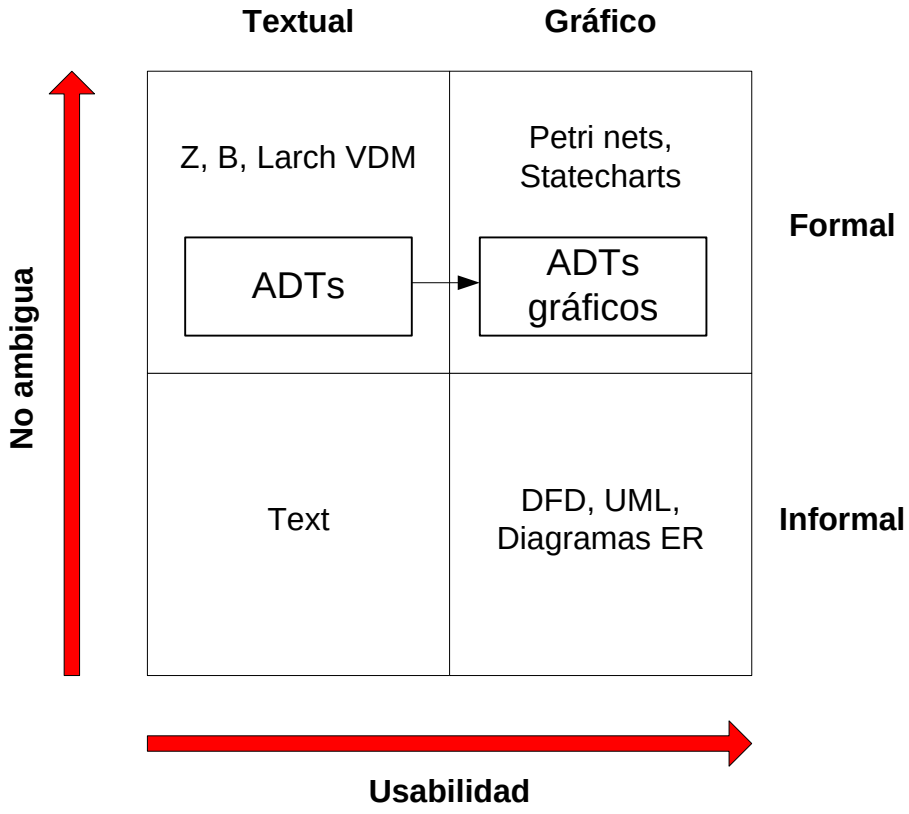


Figura 23: Clasificación de especificaciones de requerimientos

Hay dos formas extremas de especificar requerimientos de software: completa e incompleta, ejemplos de ellas se muestran en la tabla de abajo.

Informal	Semi-formal	Formal

Estas técnicas no tienen un conjunto completo de reglas que restrinjan los modelos que pueden ser creados.	Estas técnicas tienen una sintaxis definida. Instancias típicas son las técnicas de diagramación con reglas precisas que especifican las condiciones bajo las cuales la construcción es permitida, descripciones textuales y gráficas con facilidades de verificación limitadas.	Estas técnicas tienen una sintaxis y semántica rigurosamente definidas. Existe un modelo teórico mediante el cual una descripción expresada en notación matemática puede ser verificada. Los lenguajes de especificación basados en lógica predictiva son instancias típicas.
Ejemplos		
Especificaciones en lenguaje natural.	Diagramas de flujo de control, diagramas ER, UML, Use Case Diagrams	Petri nets, State Machines, ADV-charts, VDM, Z

Como se mencionó, para lograr el objetivo de mejorar y formalizar las especificaciones de patrones Web se necesita un lenguaje formal para especificar. Un lenguaje de especificación formal provee una notación (sintaxis), un universo de objetos (semántica) y la definición de reglas precisas acerca de cuáles objetos satisfacen la especificación.

Para eso, en este trabajo, se completan las especificaciones de patrones de diseño Web utilizando diagramas ADV y ADV-Charts (definidos en el capítulo anterior), y detallarán el uso de los mismos en la práctica mostrando varios ejemplos, enfocando en la interface de usuario y el comportamiento de patrones Web usados en sitios conocidos.

5.4.1 Ejemplo considerando el patrón Breadcrumbs

Para el patrón Web de ejemplo: *Breadcrumbs (indicado en la definición),* podemos ver el ADV-Chart generado.

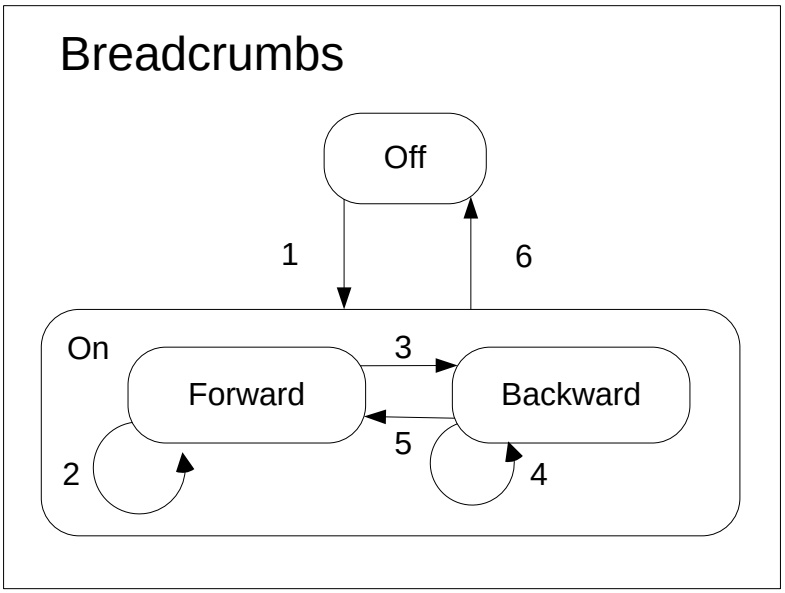


Figura 24: Diagrama ADV-Charts para el Patrón de diseño web: Breadcrumbs.

En la figura 24 se pueden ver dos estados: Off y On, los cuales indican si el componente Breadcrumbs se encuentra activado o no.
El estado On tiene dos sub-estados: Forward y Backward. El componente se encuentra en estado de navegación hacia adelante o hacia atrás respectivamente.
Donde existen las siguientes transiciones:

1:
Event: navigationLink.click()
Pre-cond: Focus(navigationLink) and homePage.visible()
Post-cond: perCont = perCont + breadCrumbs.addBreadcrumb(navigationLink) and
perCont = perCont + showContent(navigationLink)

Observación: cuando la página visible es la principal (significa que la navegación no ha comenzado) y se muestra el contenido del link navegado, el componente se activa agregando al Breadcrumbs una nueva marca con el link visitado.

2:
Event: navigationLink.click()
Pre-cond: Focus(navigationLink) and breadCrumbs.visible() and breadCrumbs.direction == FORWARD
Post-cond: breadCrumbs.addBreadcrumb(navigationLink) and perCont = perCont + showContent(navigationLink)

Observación: cuando el componente BreadCrumbs se encuentra activado (y su navegación es hacia adelante); si se presiona otro link de navegación entonces se muestra el contenido del link navegado y se agrega al componente Breadcrumbs una nueva marca con el link visitado.

3:
Event: breadCrumbs.click()
Pre-cond: Focus(navigationLink) and breadCrumbs.direction == FORWARD
Post-cond: perCont = perCont + breadCrumbs. removeBreadcrumbFrom(breadcrumbs (breadCrumbs .selected)) and perCont = perCont + showContent (breadCrumbs(breadCrumbs .selected).navigationLink)

Observación: cuando luego de navegar hacia adelante se hace click sobre el componente BreadCrumbs entonces se muestra el contenido del link guardado en la marca seleccionada del componente BreadCrumbs. También se borran todas las marcas posteriores a la marca seleccionada (inclusive esta última).

4:
Event: breadCrumbs.click() Pre-cond: Focus (breadCrumbs) and breadCrumbs.direction == BACKWARD Post-cond: perCont = perCont + breadCrumbs.removeBreadcrumbFrom breadCrumbs(breadCrumbs .selected)) and perCont = perCont + showContent(homePage)

Observación: cuando se hace click sobre uno de los Tabs del componente breadCrumbs (previamente teniendo foco sobre el mismo) se elimina del

componente breadCrumbs todos los Tabs desde el seleccionado hasta el navegado más recientemente y se muestra el contenido del link correspondiente al Tab del componente breadCrumbs que ha sido seleccionado.

5:
Event: navigationLink.click()
Pre-cond: Focus(navigationLink) and breadCrumbs.visible() and breadCrumbs.direction == BACKWARD
Post-cond: breadCrumbs.addBreadcrumb(navigationLink) and perCont = perCont + showContent(navigationLink)

Observación: cuando el componente BreadCrumbs se encuentra activado (y su navegación es hacia atrás); si se presiona otro link de navegación entonces se muestra el contenido del link navegado y se agrega al componente Breadcrumbs una nueva marca con el link visitado.

6:
Event: homePage.click()
Pre-cond: Focus(navigationLink) and breadCrumbs.visible()
Post-cond: perCont = perCont - breadCrumbs and perCont = perCont + showContent(homePage)

Observación: cuando el componente BreadCrumbs se encuentra visible y navegamos hacia la pagina principal (o fuera del sector de navegación marcado con breadcrumbs) entonces el componente se remueve del contenido y la página “principal” es mostrada.

5.5 Ejemplos de patrones Web utilizados en sitios conocidos

En esta sección se mostrarán especificaciones de algunos patrones Web encontrados en sitios conocidos. A las especificaciones se agregaron dos

secciones más: ADVs y ADV-Charts, para mostrar la interface de usuario y su comportamiento respectivamente.

5.5.1 Menú de links navegables relacionados a un elemento

Nombre / Clasificación

Floating Element menú / Navegacional.

Resumen del problema

Para determinados elementos de la pantalla es necesario ofrecer al usuario posibles links de navegación hacia otras secciones relacionadas con dicho elemento.

¿Cuándo usar?

Utilizar cuando la información a mostrar es dinámica en función del elemento mostrado en ese momento.
El tamaño real disponible en la pantalla no es suficiente para mostrar todos los links asociados al elemento.

Solución... ¿Cómo?

Cuando el usuario detiene el mouse encima del elemento sobre el cual se requiere mostrar los links de navegación, se debe desplegar una ventana que muestre todos los links y demás componentes de navegación (controles de búsqueda, etc.)

Justificación... ¿Porqué?

Si no existe suficiente espacio en la pantalla y los elementos que se muestran son dinámicos, los links de navegación no pueden mostrarse estáticamente en la misma.

Casos especiales

El menú siempre aparece de tal manera que no tapa la imagen del elemento.

Patrones relacionados

Menú “fly-out”, Menú retractable, Overlay menu.

Ejemplos

Figura 25 (a): Ejemplo del componente menú navegacional.

La figura 25 (a) (del sitio www.amazon.com) muestra una ayuda sobre la imagen de un libro que muestra una serie de links donde poder ir.

Figura 25 (b): Ejemplo del componente menú navegacional.

Como se puede ver en la figura 25 (b) se tiene incluso un campo de texto para hacer búsquedas “dentro del libro”

Diagrama ADV

En la figura 26 se muestra el diagrama ADV correspondiente a este patrón, donde se muestran posibles links de navegación a partir de un producto determinado.

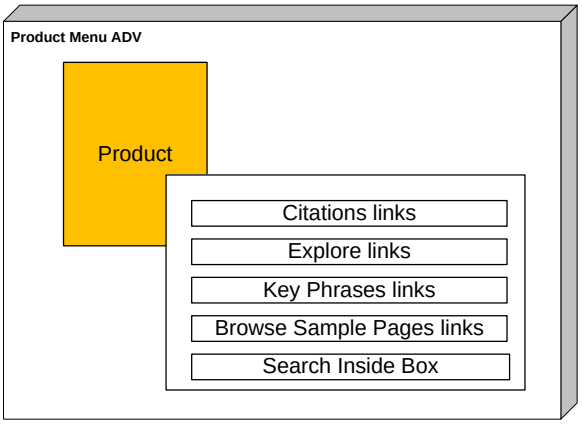


Figura 26: Diagrama ADV del menú navegacional.

ADV-Chart

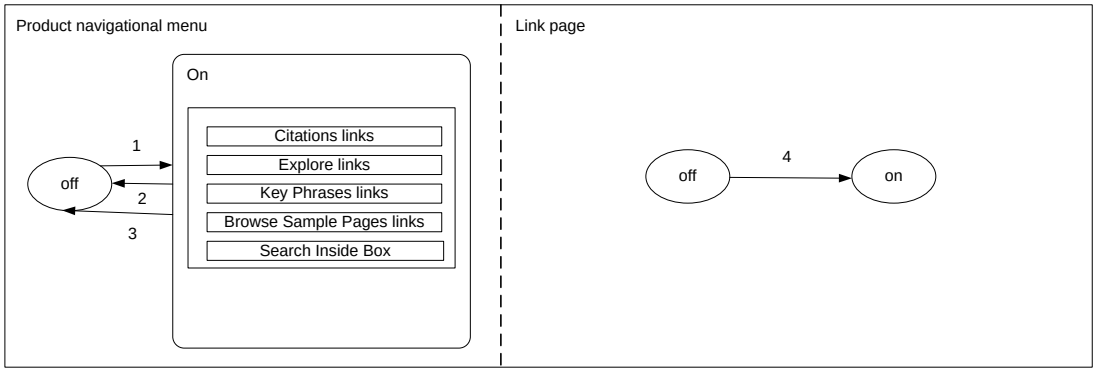


Figura 27: Diagrama ADV Chart del menú navegacional.

En la figura 27 se muestra el diagrama ADV Chart del componente, cuyos eventos se especifican a continuación:

1:
Event: MouseOver
Pre-cond: MouserOver for N time
Post-cond: ProductMenu.show
2:
Event: MouseOut
Pre-cond: ProductMenu.isOpened
Post-cond: ProductMenu.close
3:
Event: MouseClick
Pre-cond: Focus(productLink(i))
Post-cond: openLink(productLink(i).url)
4:
Event: openLink(url)
Pre-cond:
Post-cond: Page.content = url.getHTML();
perCont = perCont + Page.content

5.5.2 Calesita (Carrousel)

Nombre / Clasificación

Carrousel / Selección.

Sinónimos

Menú rotador

Resumen del problema

El problema de diseño a resolver es que los usuarios necesitan seleccionar un ítem a partir de un conjunto de ítems.

¿Cuándo usar?

Usar cuando el modelo mental de un usuario acerca de un tópico o un conjunto de objetos es muy cercano a las imágenes, como una visión gráfica del objeto; y el espacio real en la pantalla es demasiado pequeño para acomodar todas las imágenes.

Solución... ¿Cómo?

Mostrar los objetos por su representación visual de una manera circular de manera que un objeto puede ser seleccionado por vez y pueda desplazarse el subconjunto de objetos visibles en un determinado momento.
Se deben alinear todos los objetos horizontalmente o verticalmente de manera que un vector de objetos sea creado. Entonces mostrar algunos objetos del vector y ocultar el resto.
Mostrar dos flechas a los extremos del vector de manera que el usuario pueda mover el área visible. Remarcar el objeto que se encuentra seleccionado. Usar

alguna animación para lograr que el efecto de moverse por el vector se vea agradable.

Justificación... ¿Porqué?

Debido a que el Carrousel solo muestra algunos objetos en un determinado momento, solo es necesario un pequeño espacio para mostrar muchos objetos. Debido a que el Carrousel es circular, los usuarios pueden acceder a los objetos navegando hacia la izquierda o hacia la derecha.

Casos especiales

No usar el Carrousel cuando todos los objetos caben en el espacio asignado de la pantalla.
Una alternativa para mostrar variedad es presentar un conjunto diferente de objetos cada vez que la página se refresca.
El Carrousel no es adecuado para comparaciones entre objetos, especialmente cuando todos los objetos no pueden mostrarse al mismo tiempo.

Patrones relacionados

Slide transition (transición de diapositivas)
Animate (animación)

Ejemplos

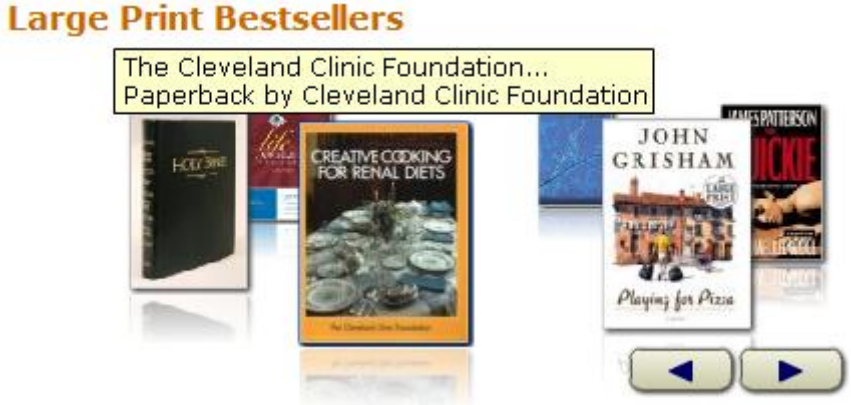


Figura 28: Ejemplo de componente Carrousel.

La figura 28 (del sitio www.amazon.com) muestra libros ordenados con este patrón.

Figura 29: Ejemplo de componente Carrousel.

La figura 29 (del sitio www.ign.com) muestra una galería de videojuegos.

Figura 30: Ejemplo de componente Carrousel.

La figura 30 (del sitio www.mupebna.com) muestra una galería de fotos de las cabañas de un complejo turístico. El componente Carrousel cuando uno posiciona el mouse amplia la foto. Continuamente se desplaza a derecha o izquierda, la dirección se puede modificar desplazando el mouse sobre el componente en el sentido inverso al que se encuentra desplazándose. Dicha funcionalidad no está explicada en nuestra especificación, por lo cual deberíamos modificar la misma en caso de querer incluirla en nuestro Carrousel.

Diagrama ADV

En la figura 31 se muestra el diagrama ADV correspondiente a este patrón, donde se muestran diferentes ítems que son ordenados circularmente para ser navegados hacia izquierda o derecha.

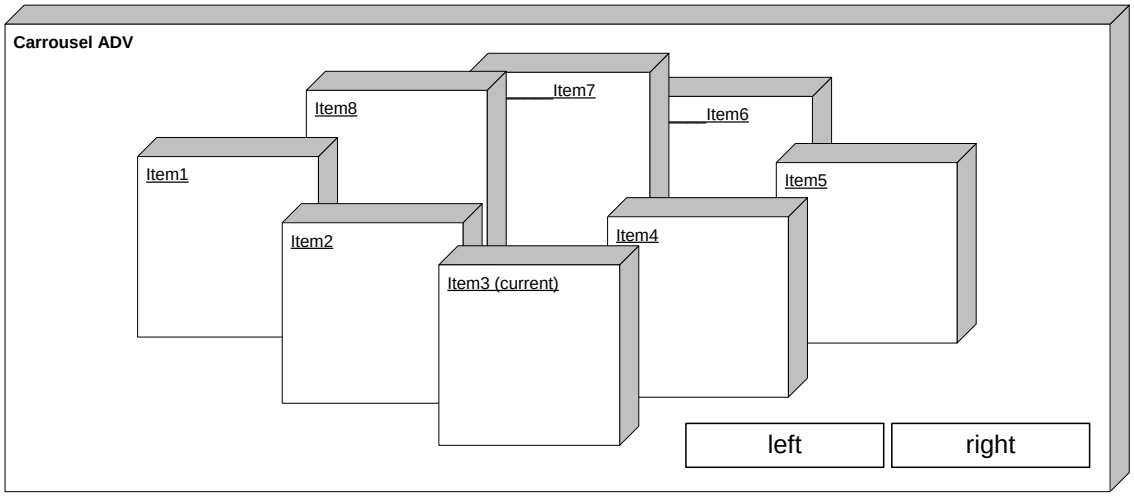


Figura 31: Diagrama ADV del Carrousel.

ADV-Chart

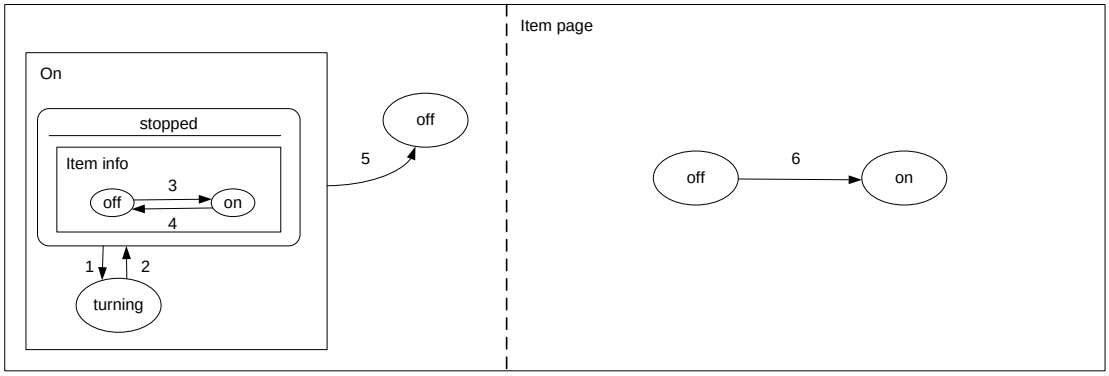


Figura 32: Diagrama ADV Chart del Carrousel.

En la figura 32 se muestra el diagrama ADV Chart del componente, cuyos eventos se especifican a continuación:

1:
Event: TurnLeftClick / TurnRightClick
Pre-cond: Focus(turnLeft/turnRight)
Post-cond: carrusel.moveLeft() / carrusel.moveRight();
2:
Event: MouseUp
Pre-cond: Focus(turnLeft/turnRight); carrousel.isTurning==true
Post-cond: carrousel.isTurning==false; perCont = perCont – items(left/right); perCont = perCont + items(right/left);
3:
Event: MouseOver
Pre-cond: Focus(item(i))
Post-cond: perCont = perCont + item(i).getInfo()
4:
Event: MouseOut
Pre-cond: Focus(item(i))
Post-cond: perCont = perCont – item(i).getInfo()
5:
Event: MouseClick
Pre-cond: Focus(item(i))
Post-cond: itemPage(i).isOpenned()

6:
Event: openPage(url)
Pre-cond:
Post-cond: Page.content = url.getHTML()

5.5.3 Calificación (Rating)

Nombre / Clasificación

Rating / Seleccionar opciones.

Sinónimos

Rating an object

Resumen del problema

El usuario quiere calificar un objeto en base a su conocimiento sobre el mismo.

¿Cuándo usar?

En cualquier sitio que trata con objetos que son ofrecidos en el sitio y donde los usuarios discuten, usan o compran los objetos y los usuarios quieren dejar una opinión rapidamente. De alguna manera u otra, cuando un sitio tiene un aspecto social, el componente Rating puede ser aplicado.

Solución... ¿Cómo?

Mostrar un componente de calificación (Rating) cerca del producto y una opción para calificarlo. Generalmente se utilizan estrellas, que se iluminan cuando pasamos el mouse sobre ellas, para inferir que pueden seleccionarse. Cuando el mouse pasa por encima del componente el nivel de calificación es indicado (cambiando el color, etc.) y se muestra un texto del nivel (Excelente, etc.)

Usualmente es ubicado al costado o abajo del producto.
Cuando el usuario selecciona su calificación, la misma debe ser guardada, el cambio dinámico en el componente debe ser congelado y un texto debe indicar que ha sido guardado.
Los usuarios deben poder cambiar en el futuro su calificación sobre un producto y guardarla nuevamente.

Justificación... ¿Porqué?

Las calificaciones están a menudo ligadas con las revisiones, las cuales incrementan las contribuciones y actividades del usuario.

Casos especiales

Hay sitios que solo permiten realizar calificaciones al administrador a los usuarios registrados.

Patrones relacionados

Calificación y revisiones (Rating and Reviews)
Votar para promover (Vote to promote)

Ejemplos

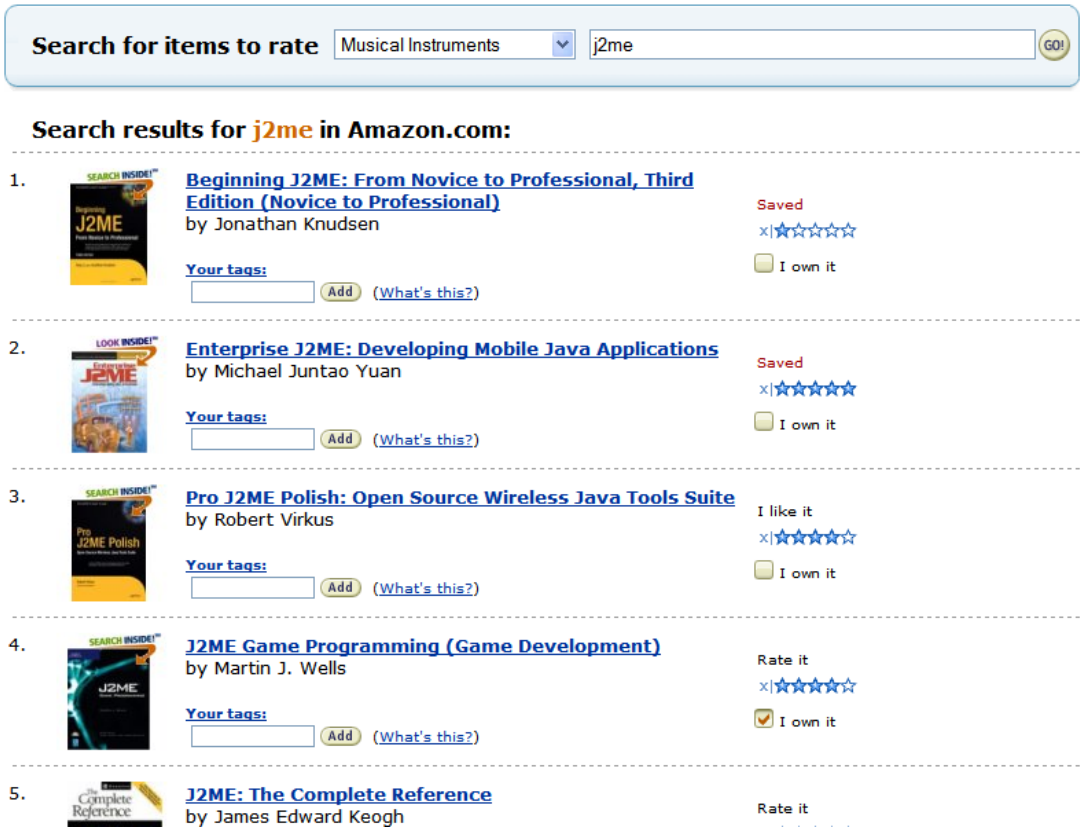


Figura 33: Ejemplo de componente Rating.

La figura 33 (del sitio www.amazon.com) muestra un listado de libros con un link a su detalle y además, en la parte derecha, un componente gráfico para calificarlo. El mismo tiene forma de cinco estrellas donde el usuario puede hacer clic en alguna de ellas para reflejar su votación.

Diagrama ADV

En la figura 34 se muestra el diagrama ADV correspondiente a este patrón, donde se muestran diferentes ítems listados para ser calificados.

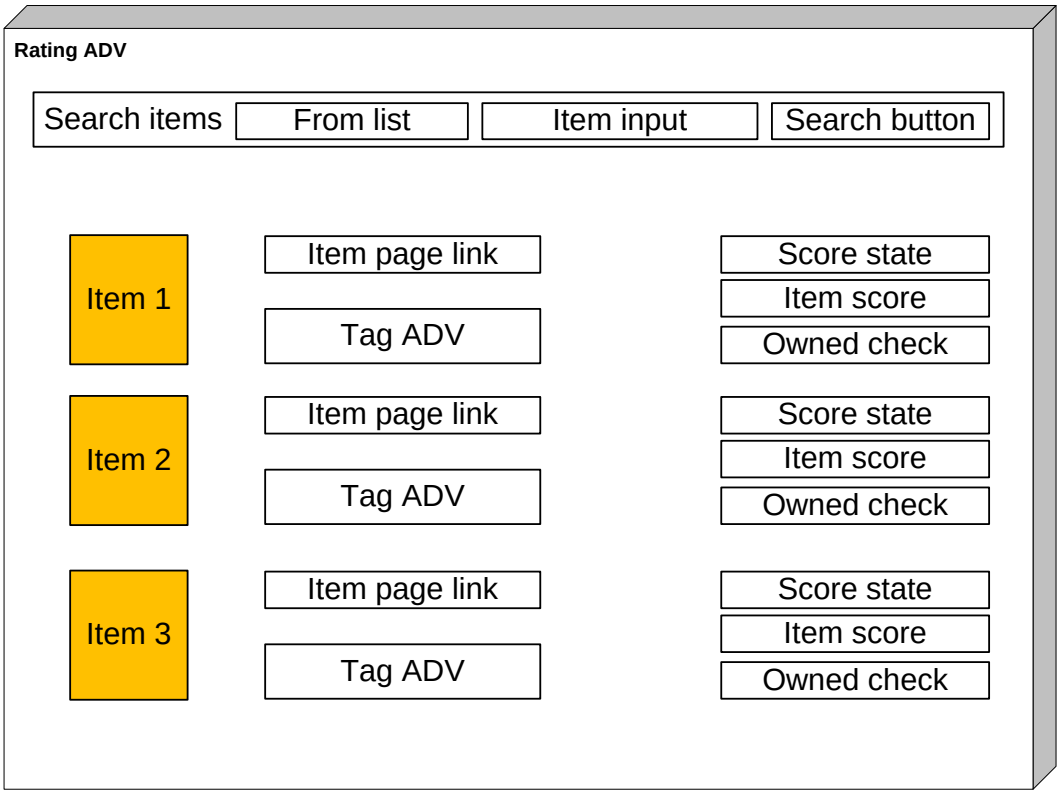


Figura 34: Diagrama ADV del Calificador.

ADV-Chart

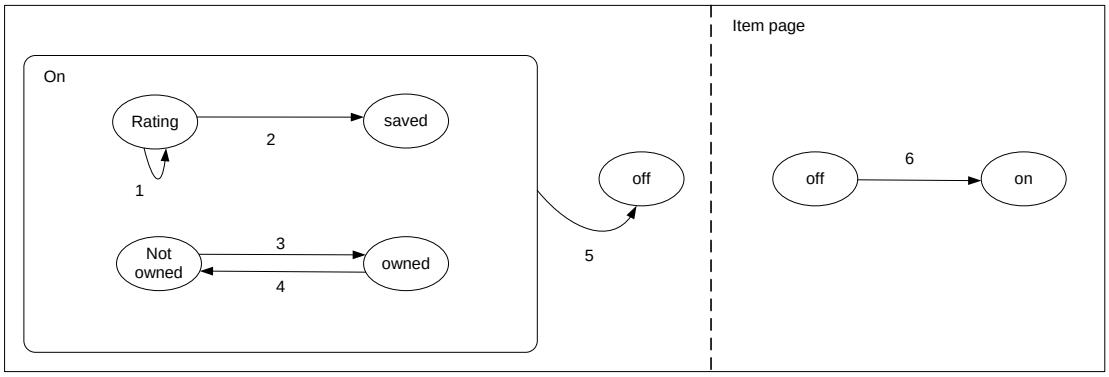


Figura 36: Diagrama ADV Chart del Calificador.

En la figura 36 se muestra el diagrama ADV Chart del componente, cuyos eventos se especifican a continuación:

1:

Event: MouseOver

Pre-cond: Focus(starsImage(i))

Post-cond: item(i).showScoreDescription

2:

Event: MouseClick

Pre-cond: Focus(starsImage(i))

Post-cond: item(i).saveScore

3:

Event: Check

Pre-cond: Focus(ownedCheck(i))

Post-cond: item(i).owned

4:

Event: Uncheck

Pre-cond: Focus(ownedCheck(i))

Post-cond: item(i).notOwned

5:

Event: MouseClick

Pre-cond: Focus(item(i))

Post-cond: openLink(item(i).url)

6:

Event: openLink(url)

Pre-cond:

Post-cond: Page.content = url.getHTML();

perCont = perCont + Page.content

5.6 Ejemplo de aplicación Web diseñada y especificada utilizando OOHDM

En el siguiente ejemplo se muestran los cambios realizados a nivel de interface de usuario Web (comportamiento y estado) cuando se transforma una aplicación Web Tradicional en una aplicación Web RIA.

Para esto tomamos como ejemplo una aplicación existente que permite accederla de ambas maneras: tradicional y RIA.

Cuando se ingresa al sitio de Amazon (www.amazon.com) con un usuario registrado, podemos acceder a la sección de recomendaciones personalizadas.

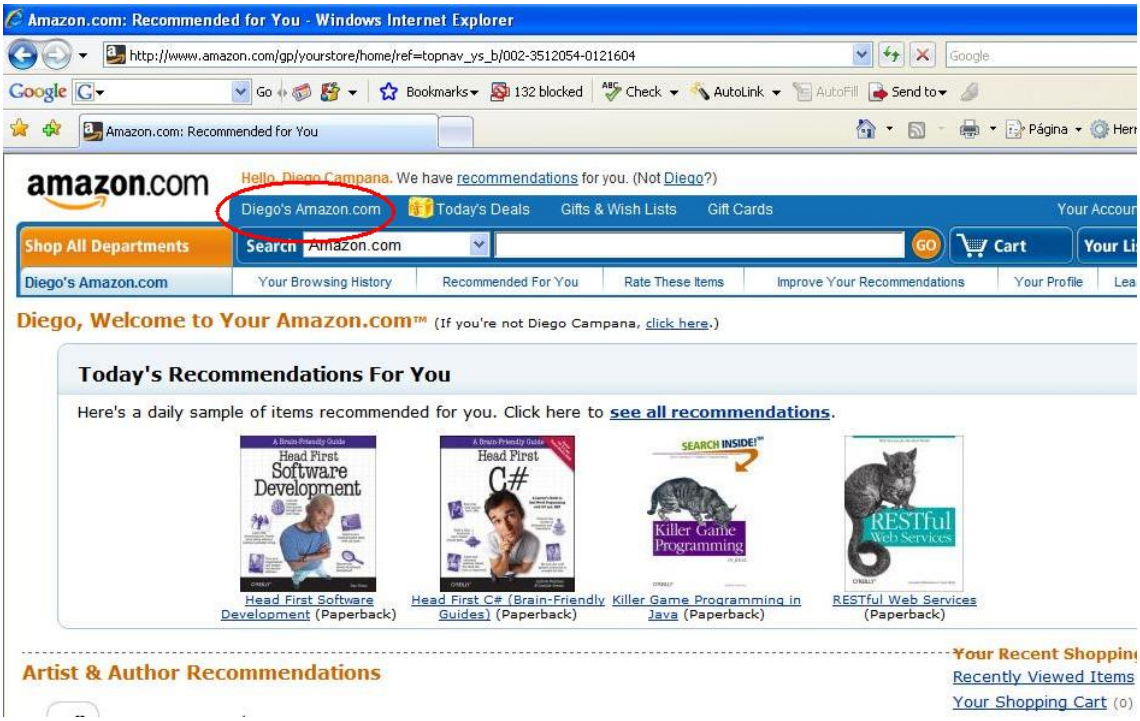


Figura 37 (a): Sección del sitio web Amazon.

En la figura 37 (a) se muestra la pantalla del sitio cuando se navega a la sección personalizada desde el menú (indicado con un círculo rojo). Esta pantalla se llamará “Recomendaciones de hoy”.

Navegando la pantalla hacia abajo como muestra la figura 37 (b) esta la sección de Artistas y Autores Recomendados. Este ejemplo tomará solamente la posibilidad de Libros y Autores. En la misma se ve la descripción del libro, el autor, el precio del libro nuevo, el mínimo precio usado, el “rating” actual, y el motivo de la recomendación, así como links para navegar a cada una de estas opciones. Para el propósito de este trabajo a esta sección de la pantalla se llamará *“Recomendaciones de Autores”.*

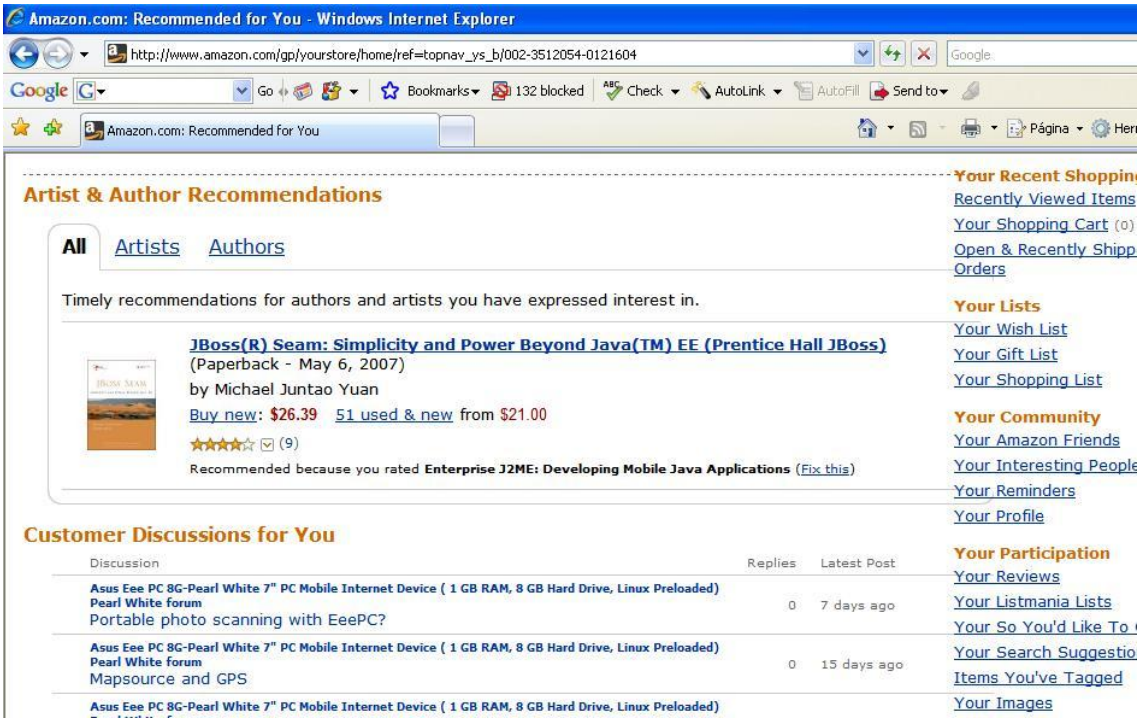


Figura 37 (b): Sección del sitio web Amazon.

5.6.1 Análisis del diseño con OOHDM

Para mostrar el cambio efectuado en la capa de interface de usuario al cambiar el acceso de tradicional a RIA, se realizó un análisis simplificado de la aplicación para mostrar el modelo de negocio que la sustenta, su vinculación con la capa de navegación y su posterior acceso desde las vistas, como se puede ver en la figura 38.

La capa conceptual es un supuesto entre los posibles modelos de negocio. Así como también hay información de la capa de navegación que no se incluyo por simplicidad, como ser el diagrama de estructuras de acceso.

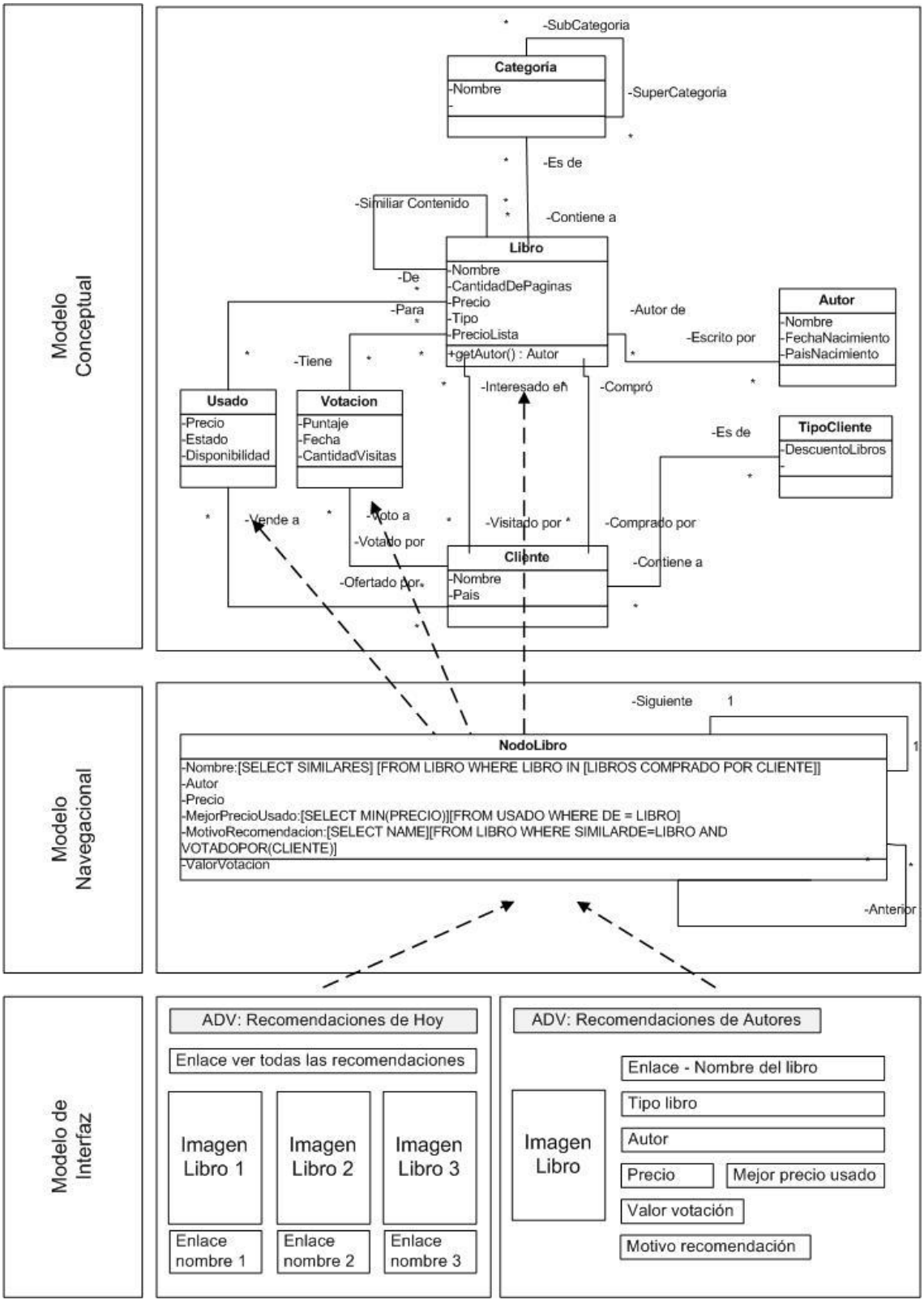


Figura 38: Diagrama de capas del modelo OOHDM.

5.6.2 Aplicación Web RIA

Para ver la transformación de la sección de recomendaciones de Tradicional a RIA, simplemente se habilita la posibilidad de ejecutar javascript en el navegador cliente.

Ingresando nuevamente a la sección se puede ver el cambio de la aplicación.

En la figura 39 (a) se ve el cambio de componente para mostrar los libros recomendados. Ahora se utiliza un “Carrusel”, cuyo patrón y especificación (con ADVs – ADV Charts) fue mostrado anteriormente aquí.

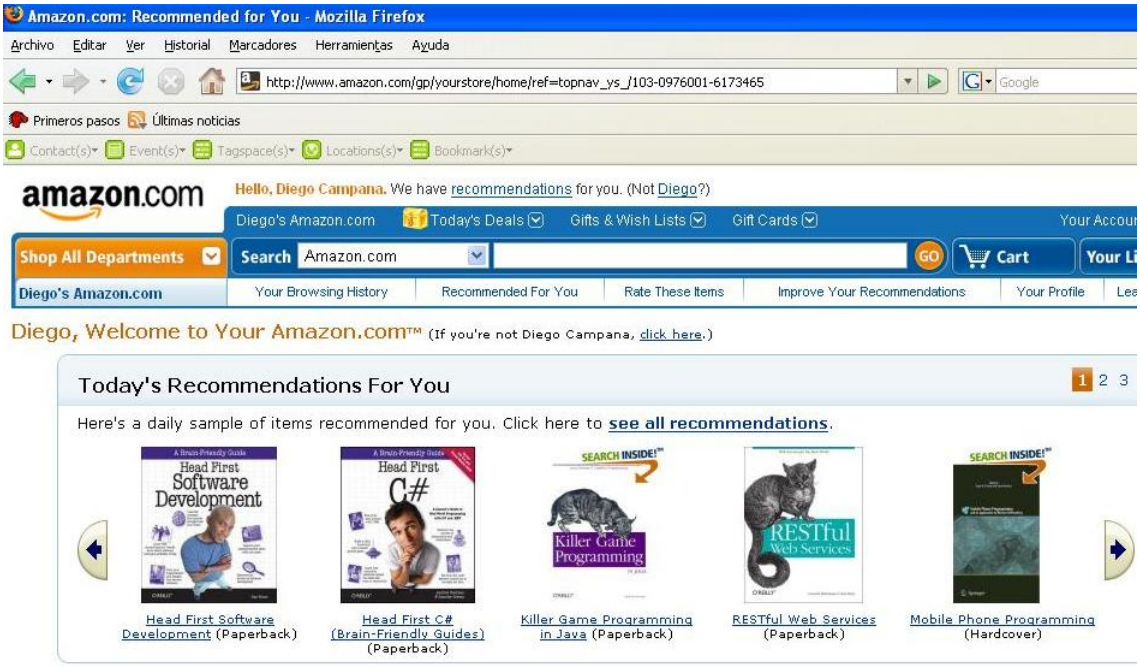


Figura 39 (a): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

La figura 39 (b) muestra el menú navegacional, desplegado cuando el usuario se detiene sobre una de las imágenes de libros. Este patrón fue abordado anteriormente aquí.

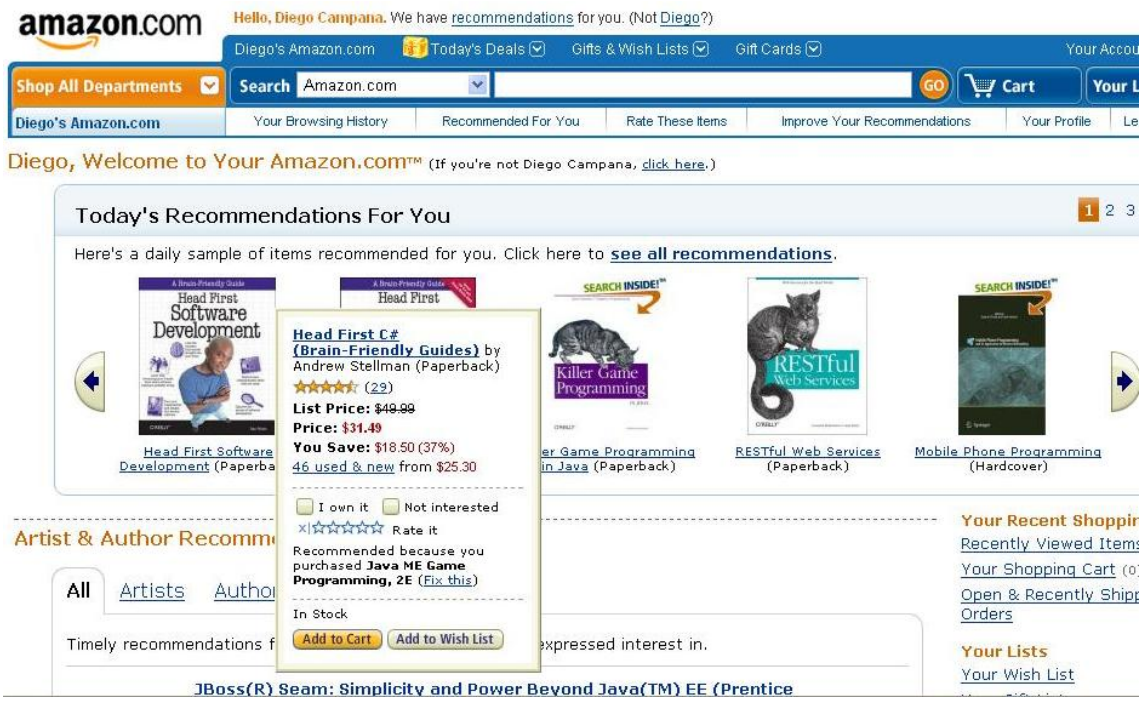


Figura 39 (b): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

En la figura 39 © se muestra la sección: “Recomendación de Autores” uno de los cambios es la inclusión del “Menú Navegacional” del libro (tratado arriba)

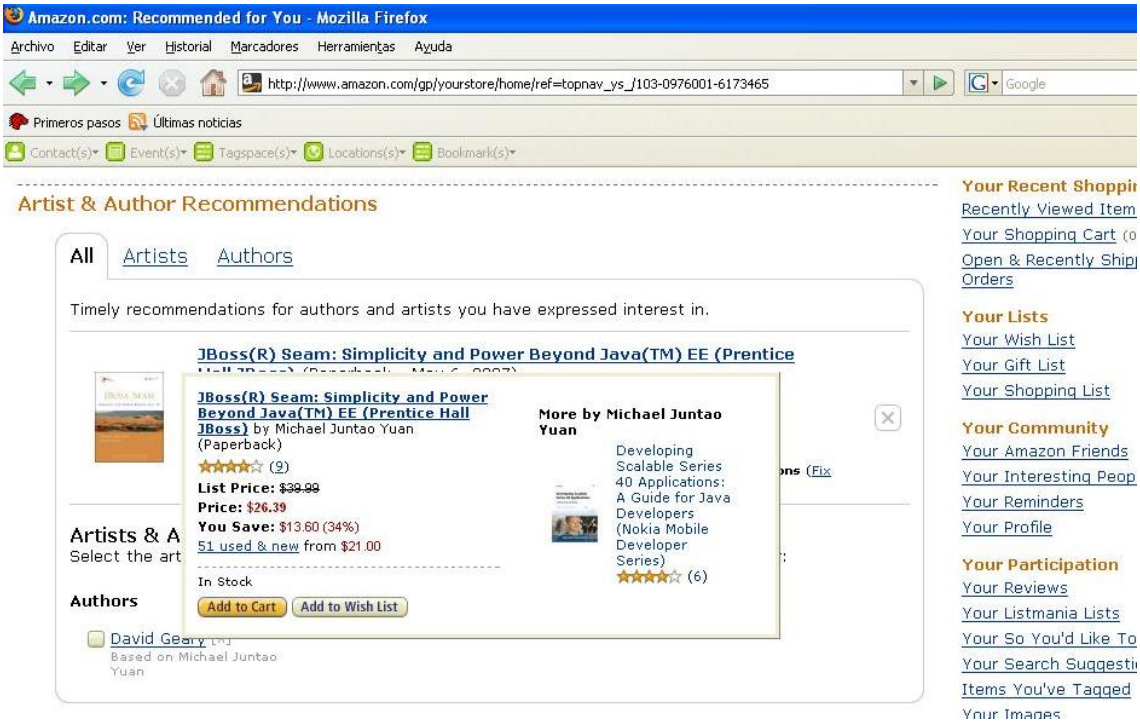


Figura 39 (c): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

También como parte del mismo patrón, un detalle del origen del "rating" es mostrado cuando se pasa el Mouse sobre las estrellas que indican el puntaje, permitiendo ir a los detalles de las votaciones, como muestra la figura 39 (d).

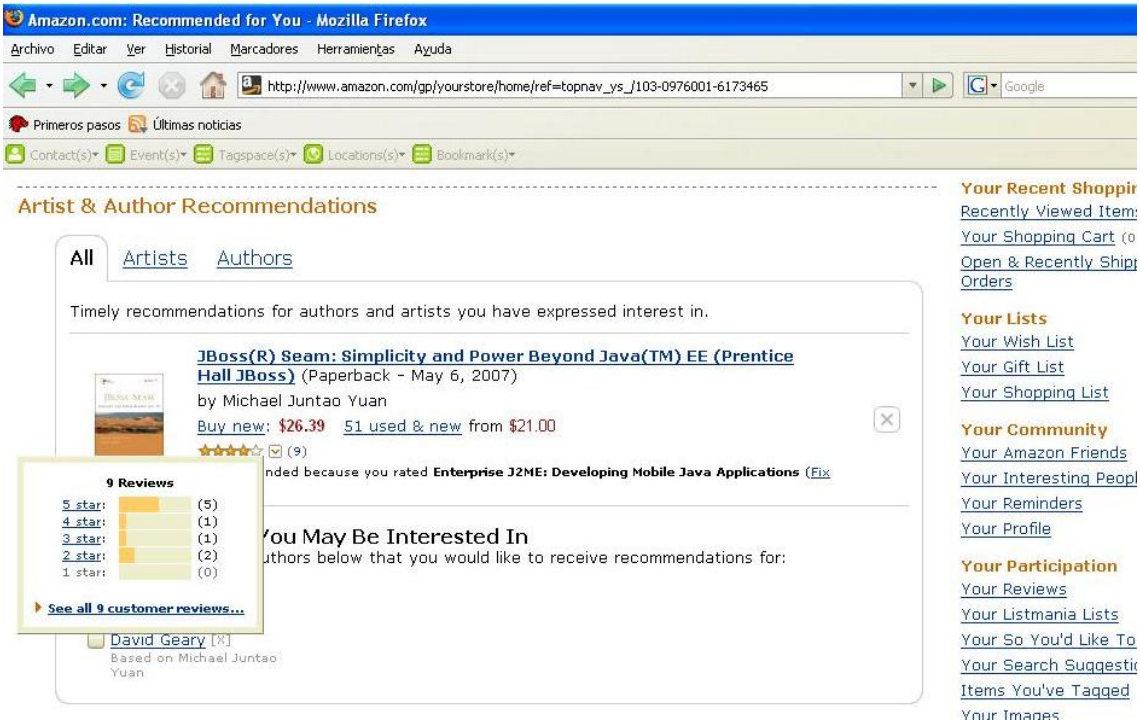


Figura 39 (d): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

El siguiente patrón Web (cuadro color rojo) es incluido para tratar la elección de los autores recomendados, permitiendo seleccionar aquellos autores en los cuales se está interesado y asignar una puntuación a los mismos, dicho comportamiento se muestra en la figura 39 (e).

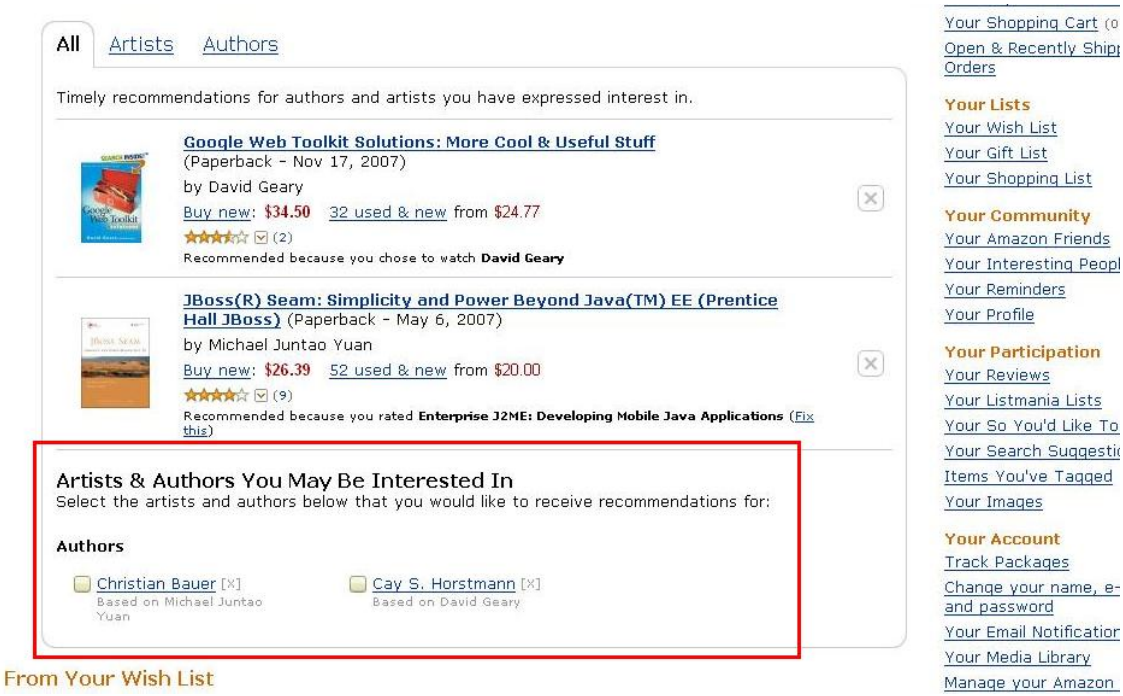


Figura 39 (e): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

La figura 39 (f) muestra lo que ocurre cuando es seleccionado un autor de los propuestos: un mensaje explicativo de la acción es mostrado arriba, y la etiqueta “saved” es mostrada.

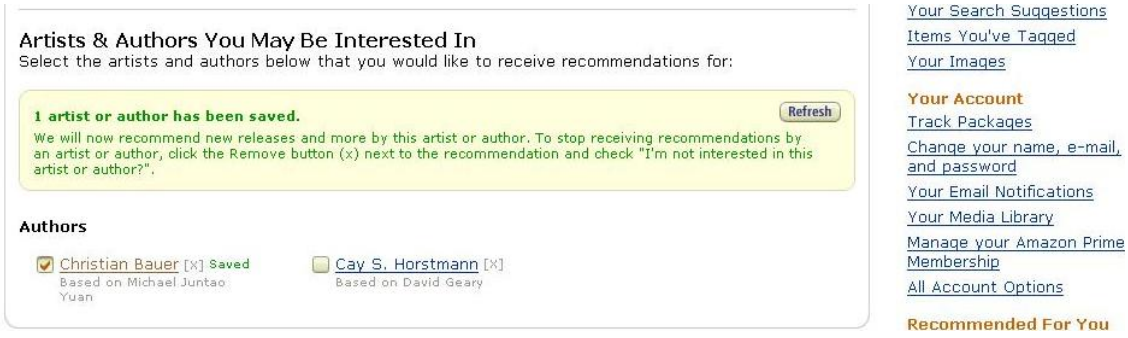


Figura 39 (f): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

También existe la posibilidad de eliminarlo como propuesto, presionando la “cruz” que se encuentra a la derecha del nombre, lo cual se puede ver en la figura 39 (g).

Figura 39 (g): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

Este componente no tiene un equivalente en la versión tradicional para la sección de autores recomendados (como puede verse en la figura 39 (h)).

Figura 39 (h): Sección del sitio web Amazon con RIA (javascript habilitado en nuestro navegador).

En la figura 40 se muestra el ADV Chart correspondiente a este componente Web (para seleccionar autores recomendados).

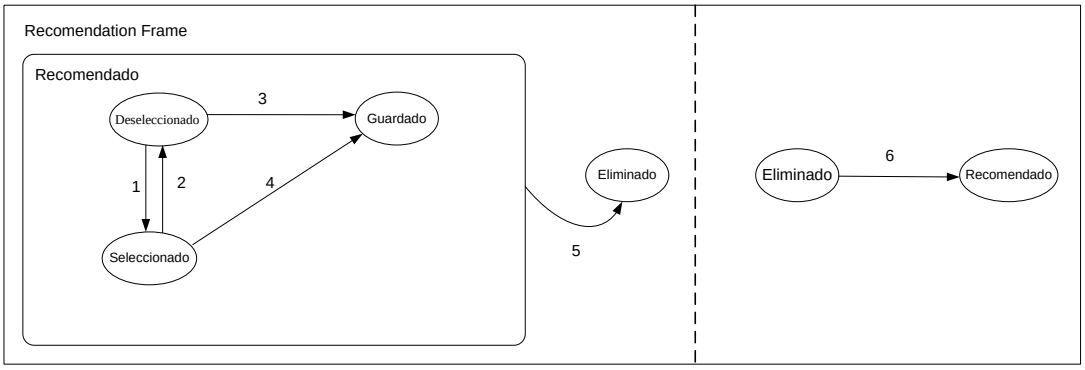


Figura 40: Diagrama ADV Chart para componente RIA de selección de autores recomendados.

1:

Event: MouseClick

Pre-cond: Focus(checkbox(i))

Post-cond: author(i).showRecomendationAdvise

2:

Event: MouseClick

Pre-cond: Focus(checkbox(i))

Post-cond: author(i).hideRecomendationAdvise

3:

Event: MouseClick

Pre-cond: Focus(checkbox(i))

Post-cond: author(i).showSavedLabel

4:

Event: MouseClick

Pre-cond: Focus(checkbox(i))

Post-cond: author(i).showSavedLabel

5:

Event: MouseClick

Pre-cond: Focus(crossicon(i))

Post-cond: hideAuthor(i); showUndoOption(i);

6:

Event: MouseClick

Pre-cond: Focus(undoOption(i))

Post-cond: showAuthor(i);

Capítulo 6. Evaluación de la adaptación de aplicaciones tradicionales a RIA

6.1 Aplicación tradicional que muestra un conjunto de ítems en forma gráfica

La aplicación del ejemplo tiene la necesidad de mostrar al usuario un conjunto de ítems recomendados. Para esto consulta un servicio de aplicación que muestra un conjunto de imágenes que corresponden a los ítems. El conjunto de ítems es seleccionado en base al identificador del usuario enviado como parámetro.
Para simplificar este ejemplo no se utiliza ningún framework, simplemente un HTML que invoca a un servlet cuya respuesta es el HTML con la lista de ítems.

6.1.1 Página HTML principal:

Cuando es seleccionada una de las dos opciones (ver figura 41 (a)) el formulario de la página HTML invoca a un servicio en el servidor Web que se ocupa de obtener la lista de ítems correspondientes para los parámetros enviados.

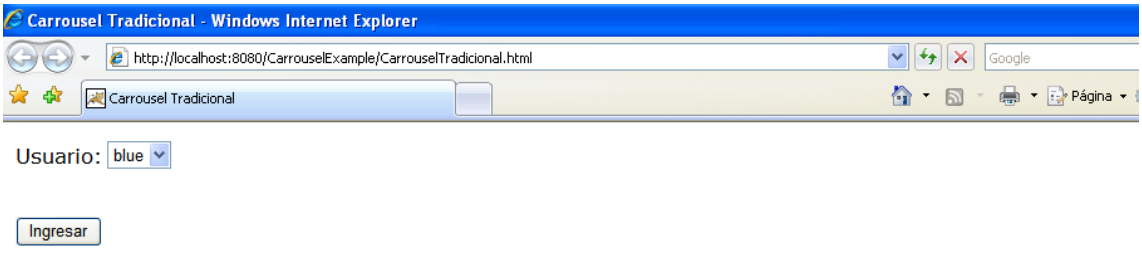


Figura 41 (a): Ejemplo aplicación web tradicional. HTML principal.

6.1.2 Código HTML:

```
<html>
```

```
<head>
        <title>Carrousel Tradicional</title>
        <meta http-equiv="content-type" content="text/html;charset=UTF-8"/>
</head>
<body>
        <form action="CarrouselServletTradicional" method="POST">
                <font face="Verdana">Usuario:</font>
                <select name="itemType">
                        <option selected="">blue</option><option>red</option>
                </select>
                <br><br><br>
                <input type="submit" value="Ingresar" ></input>
                <input type="hidden" value="5" name="itemsAmount"></input>
        </form>
</body>
</html>
```

El identificador del usuario se selecciona de una lista de opciones y los valores posibles son: “blue” o “red”.

Las imágenes que se muestran para los ítems son de color azul o rojo dependiendo de cuál es el color seleccionado.

Se toma como ejemplo el uso de dos colores para mostrar claramente la diferencia de ítems entre una opción y la otra.

El valor del elemento “itemsAmount” determina la cantidad de ítems a mostrar.

6.1.3 Método del servicio que procesa la solicitud (J2EE Servlet):

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
          /*
          Recupera parámetros y evalúa el tipo de ítems a buscar.
          */
         String itemType = request.getParameter("itemType");
         int itemsAmount = Integer.parseInt(request.getParameter("itemsAmount"));
           List items = new ArrayList();
         if (itemType != null) {
            if ("blue".equals(itemType)) {
               items = Item.getBlueItems();
```

```
            } else if ("red".equals(itemType)) {
                items = Item.getRedItems();
            }
        }
        /*
        Genera el código HTML para la página de resultado mostrando la cantidad de ítems solicitada.
        */
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Carrousel Tradicional</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<table width=\"761\" border=\"1\" height=\"131\">");
          out.println("<tr height=\"83\"><td><table width=\"747\" border=\"0\"
          height=\"124\">");
          out.println("<tr height=\"94\"><td><table height=\"92\" width=\"693\"
          border=\"0\"><tr height=\"84\">");
        for (int i = 0; i<itemsAmount;i++){
            out.println("<td width=\"74\"><img src=\"" +  ((Item)items.get(i)).image + "\"
            height=\"85\" width=\"85\" align=\"middle\" border=\"0\"></td>");
        }
        out.println("</tr></table></td></tr></table></td></tr></table>");
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}
```

En la primer parte del método (color naranja) se recuperan los parámetros y se invoca al servicio apropiado para recuperar ítems de color azul o rojo.
Este servicio devuelve una lista de ítems. Una parte del mismo es mostrado en la página de resultado (color verde).

6.1.4 Página HTML con los ítems como resultado:

El mismo servicio Web se ocupa de generar la página HTML con los ítems como resultado como muestra la figura 41 (b).

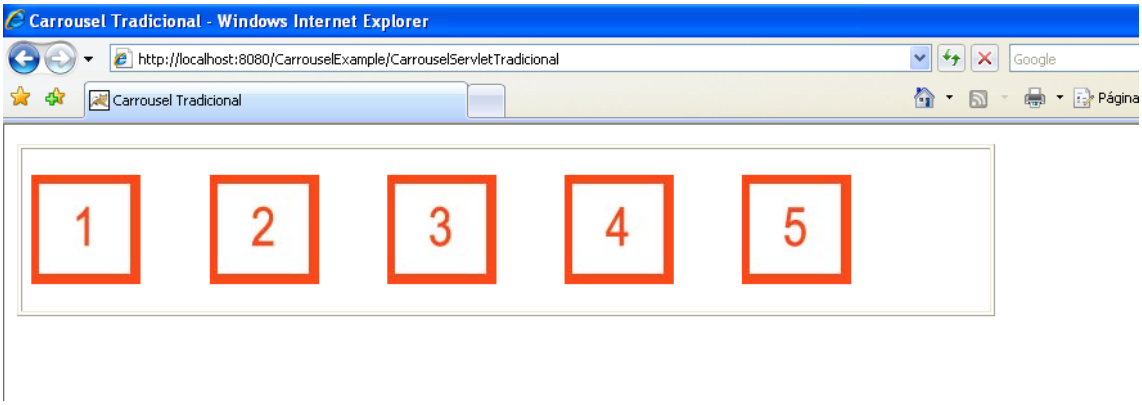


Figura 41 (b): Ejemplo aplicación web tradicional. HTML página resultado.

6.1.5 Aplicación RIA que muestra un menú de ítems en forma gráfica (Carrousel)

El patrón de diseño de la Calesita (como se dijo anteriormente en este trabajo), da solución al problema de mostrar un conjunto pequeño de ítems limitado por el espacio en la página. Esto se logra ocultando una parte del conjunto de ítems y permitiéndole al usuario recorrerlos desplazándose a izquierda o derecha sin necesidad de recargar la página nuevamente desde el servidor.
Los siguientes cambios son aplicados a los componentes de la aplicación.

6.1.6 Página HTML principal:

A la página HTML principal se agregan los siguientes elementos (ver figura 42 (a)):

- Dos elementos DIV para poder ocultar y mostrar la lista de selección y el componente gráfico “Carrousel” (en un comienzo oculto).
- Una imagen cualquiera para poder visualizar más fácilmente que la página HTML no se recarga de nuevo.
- Llamadas a los métodos de la capa RIA (AJAX) que se ocupan de buscar los ítems del servidor y mostrarlos en el componente gráfico, interactuando con el usuario para recorrerlos.

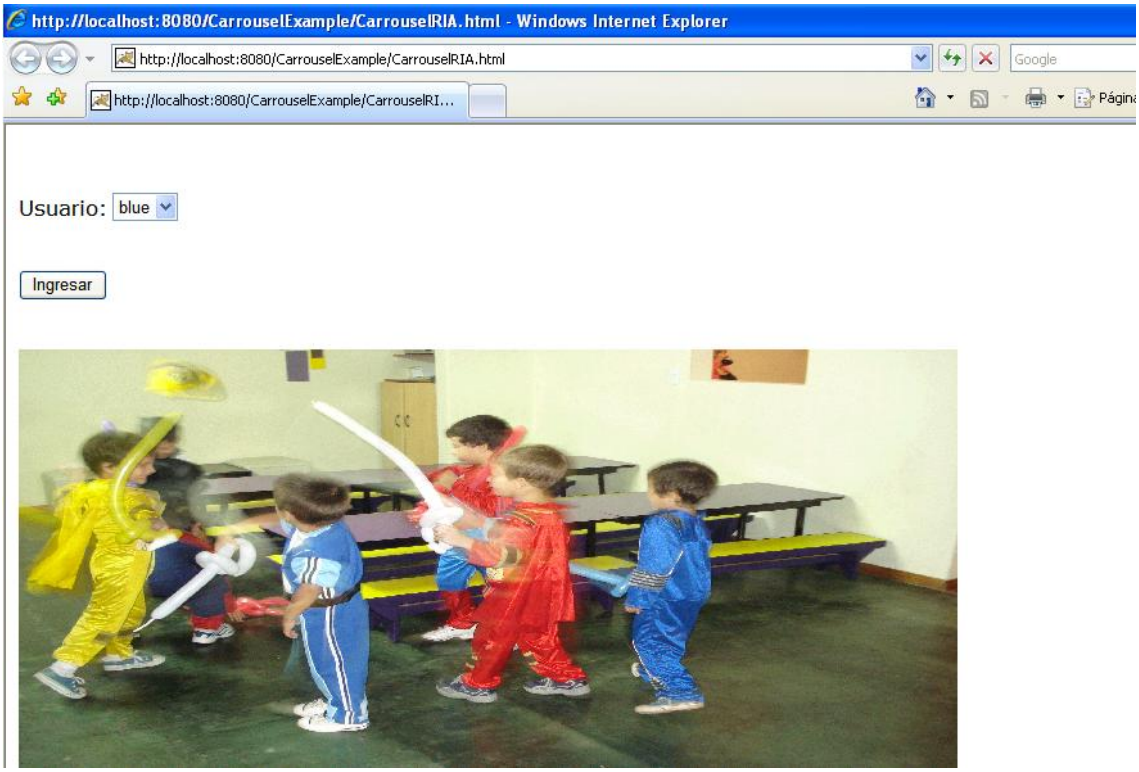


Figura 42 (a): Ejemplo aplicación web RIA. HTML principal.

6.1.7 Código HTML:

```
<head>
  <title></title>
  <meta http-equiv="content-type" content="text/html;charset=UTF-8"/>
</head>
<body>
<div id="carrouselComponent" style="DISPLAY: none">
<table width="761" border="1" height="131">
        <tr height="83"><!-- Row 1 -->
        <td>
        <table width="747" border="0" height="124">
                <tr height="94"><!-- Row 1 -->
                <td>
        <table height="92" width="693" border="0">
        <tr height="84">
                 <td width="74"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="70"><img height="85" alt="" width="85" align="middle" border="0"></td>
                 <td width="74"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="69"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="73"><img height="85" alt="" width="85" align="middle" border="0"></td>
```

```
                    <td width="81"><img height="85" alt="" width="85" align="middle" border="0"></td>
                    <td width="62"><img height="85" alt="" width="85" align="middle" border="0"></td>
                    <td><img height="85" alt="" width="85" align="middle" border="0"></td>
                    </tr>
                    </table>
                    </td>
                    </tr>
            </table>
            <img align="right" height="18" alt="" src="Imagenes/right.png" width="18" border="0"
            onclick="right();">
            <img align="right" height="18" alt="" src="Imagenes/left.png" width="18" border="0"
            onclick="left()">
            </td>
            </tr>
</table>
<br>
<input align="right" type="button" value="Home"
onclick="document.getElementById('carrouselComponent').style.display='none';
        document.getElementById('loginForm').style.display='';">
</div>
<br><br>
<div id="loginForm">
<form>
<font face="Verdana">Usuario:</font>
<select id="color"><option selected="">blue</option><option>red</option></select> <br><br><br>
<input type="button" value="Ingresar"
onclick="getItems(document.getElementById('color').options[document.getElementById('color').selectedInd
ex].text);"></input>
</form><br>
</div>
<img height="350" alt="" src="Imagenes/marb014.jpg" width="731" border="0">
</body>
</html>
```

6.1.8 Código javascript para la navegación de los ítems e interacción con la capa RIA:

```
<script type="text/javascript" language="javascript" src="ajax.js"></script>
<script type="text/javascript" language="javascript" src="carrousel.js"></script>
<script language="javascript">
var index = 0;
var amountImg = 15;
var amountVis = 8;
```

```
var loadingImage = false;
function right(){
	if (index<amountImg)
		index = index + 1;
	else
		index = 0;
	var i=0;
	var j=0;
	while (i < amountVis){
		if ((i + index) < amountImg){
			j = i + index;
		}
		else{
			j = amountImg - (i + index + 1);
			if (j < 0)
				j = j * -1;
			j = j - 1;
		}
		document.images[i].src = pics[j].src;
		i=i+1;
	}
}
function left()
{
	if (index>0)
		index = index - 1;
	else
		index = 14;
	var i=0;
	var j=0;
	while (i < amountVis){
		if ((i + index) < amountImg){
			j = i + index;
		}
		else{
			j = amountImg - (i + index + 1);
			if (j < 0)
				j = j * -1;
			j = j - 1;
		}
		document.images[i].src = pics[j].src;
		i=i+1;
	}
}
```

```
function LoadImages(){
        if (loadingImage) return;
        loadingImage = true;
      var i=0;
        while (i<amountVis){
                if (i<amountVis){
                        document.images[i].src = pics[i].src;
                }
                i=i+1;
        }
        loadingImage = false;
}
</script>
```

6.1.8.1 Carrousel.js:

En este punto es necesario tener definido el formato de los datos para el dialogo entre la capa RIA en el cliente y la capa RIA en el servidor.

De las posibilidades existentes las más utilizadas son XML y JSON. JSON representa mejor la estructura de los datos, requiriendo menos codificación y procesamiento respecto a XML.

En el caso de transformar una aplicación Web tradicional existente en una aplicación RIA, debemos por lo tanto utilizar vistas del modelo de aplicación.

La nueva vista del modelo debe permitir accederlo utilizando la nueva representación acordada entre las capas RIA del cliente y el servidor.

En esta transformación es posible que por diversos motivos sea requerido además de exponer la funcionalidad existente, crear nueva lógica de negocio.

En las siguientes secciones se muestra como debe generarse esta nueva vista modificando la representación actual de la misma. En el diagrama de secuencia resultado de la transformación Web a RIA, la clase “Assembler” correspondiente, es la responsable de traducir las representaciones.

```
var pics = new Array();
var descriptions = new Array();
function getItems(type) {
        var req = newXMLHttpRequest();
        var handlerFunction = getReadyStateHandler(req, updateItems);
        req.onreadystatechange = handlerFunction;
```

```
		req.open("POST", "CarrouselServlet", true);
		req.setRequestHeader("Content-Type",
			"application/x-www-form-urlencoded");
		req.send("itemType="+type);
}
function updateItems(itemsXML) {
		var items = itemsXML.getElementsByTagName("items")[0];
		var itemsVar = items.getElementsByTagName("item");
		for (var I = 0 ; I < itemsVar.length ; I++) {
			var item = itemsVar[I];
			var image = item.getElementsByTagName("image")[0]
				.firstChild.nodeValue;
		var information = item.getElementsByTagName("information")[0]
				.firstChild.nodeValue;
			pics[I] = new Image();
			pics[I].src = image;
			descriptions[I] = information;
		}
		document.getElementById('carrouselComponent').style.display='';
		document.getElementById('loginForm').style.display='none';
		LoadImages();
}
```

6.1.8.2 Ajax.js:

```
function newXMLHttpRequest() {
		var xmlreq = false;
		if (window.XMLHttpRequest) {
			xmlreq = new XMLHttpRequest();
		} else if (window.ActiveXObject) {
			try {
				xmlreq = new ActiveXObject("Msxml2.XMLHTTP");
			} catch (e1) {
				try {
					xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
				} catch (e2) {
					xmlreq = false;
				}
			}
		}
		return xmlreq;
}
function getReadyStateHandler(req, responseXmlHandler) {
		return function () {
```

```
                if (req.readyState == 4) {
                        if (req.status == 200) {
                                responseXmlHandler(req.responseXML);
                        } else {
                                alert("HTTP error "+req.status+": "+req.statusText);
                        }
                }
        }
}
```

6.1.9 Método del servicio que procesa la solicitud (J2EE Servlet):

El método del servidor encargado de recuperar los ítems y generar la página de resultado, fue modificado de la siguiente manera (color naranja):

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        String itemType = request.getParameter("itemType");
        List items = new ArrayList();
        if (itemType != null) {
            if ("blue".equals(itemType)) {
                items = Item.getBlueItems();
            } else if ("red".equals(itemType)) {
                items = Item.getRedItems();
            }
        }
        String itemsXml = Item.toXML(items);
        // Write XML to response.
        response.setContentType("application/xml");
        response.getWriter().write(itemsXml);
    } finally {
        out.close();
    }
}
```

La nueva sección de código hace uso de un método para convertir la lista de ítems en su representación XML y escribir dicho documento como salida.

En color naranja (en el código de arriba) se traduce de la representación de objetos a XML mediante el método “toXML” de la clase Item. Luego se selecciona el tipo de contenido de la respuesta (application/xml) y por último se envía el XML como respuesta.

Este cambio de representación no debe ser resp3onsabilidad del propio objeto, conviene tener la lógica de transformación separada, centralizada en una clase determinada. En la sección: El nuevo controlador, se presenta la clase “Assembler”, encargada de esta tarea.

La clase Item es un objeto utilizado por el cliente en la aplicación Web tradicional. En este ejemplo ella es responsable de su propia traducción a la nueva representación. Esta nueva funcionalidad fue agregada para la transformación a RIA.

En la aplicación de ejemplo se tuvo que modificar el Servlet que accede a los datos del negocio para invocar al método de traducción a XML.

En este caso el objetivo es mostrar de manera sencilla el cambio de representación, luego se verá un caso de ejemplo para transformar una aplicación Web tradicional utilizando patrones de diseño apropiados.

6.1.10 Página HTML con los ítems como resultado:

Como muestra la figura 42 (b), la misma página HTML que invocó el servicio del servidor para mostrar los ítems muestra la respuesta obtenida del servidor, utilizando para esto el componente Carrousel.

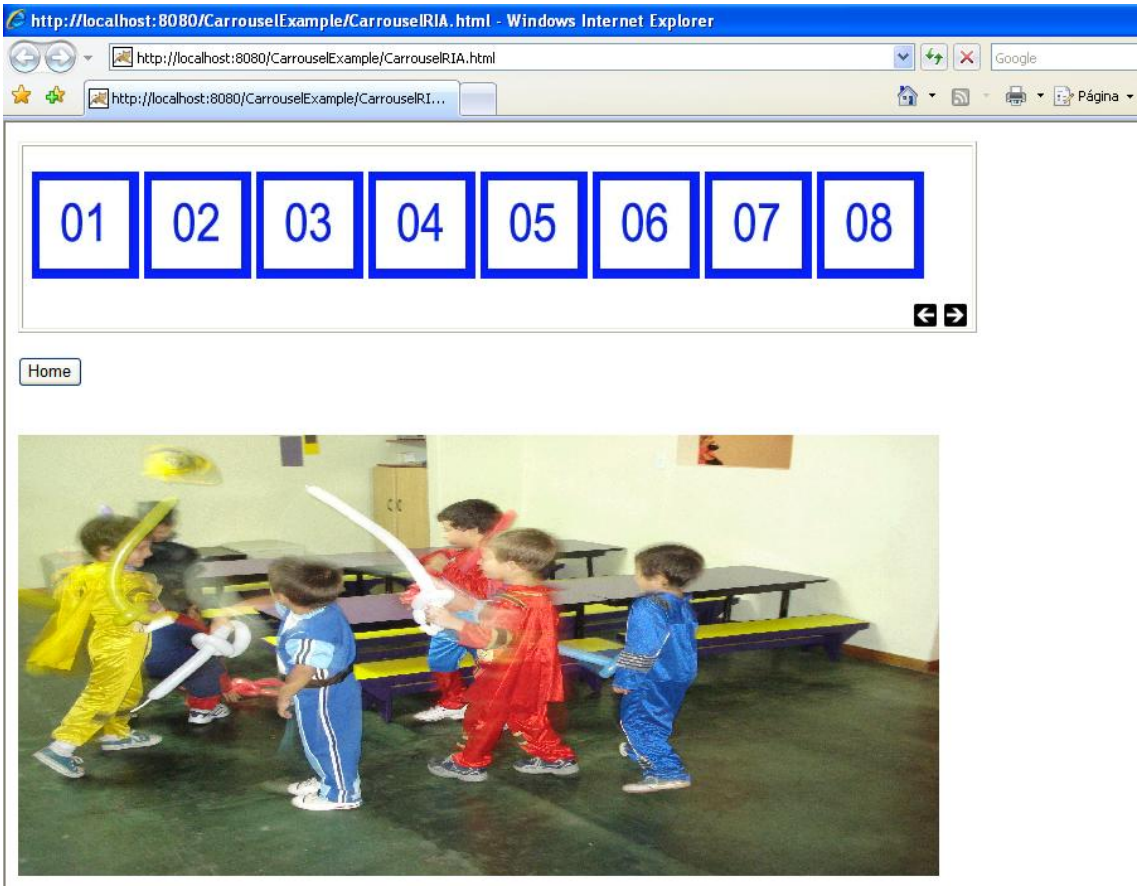


Figura 42 (b): Ejemplo aplicación web RIA. HTML principal mostrando el componente Carrousel.

Como se ve, no hay una recarga de la página, solo se oculta una parte de ella y se muestra otra parte con nueva información obtenida del servidor.

6.2 Análisis de la adaptación

En base al ejemplo expuesto, se puede ver que la adaptación de una aplicación tradicional en RIA, consta de dos modificaciones fundamentales:

1. Incorporación de una capa RIA en el cliente para comunicarse con el servidor y funcionalidad necesaria para cargar y mostrar información en el navegador.

2 Modificación del servicio de la capa de aplicación para devolver la información requerida mediante XML, en el formato preestablecido (la traducción necesaria de objetos de la capa de presentación en su representación XML).

La figura 43 muestra gráficamente donde impactan estos cambios dentro de la arquitectura.

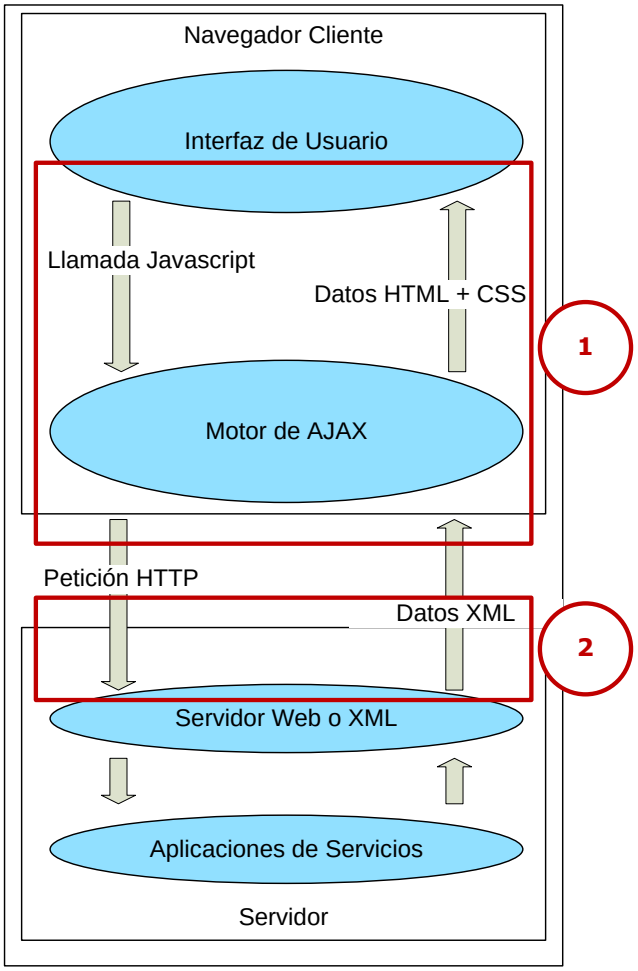


Figura 43: Modificación por capas en la adaptación a RIA.

6.2.1 Analizando los diferentes diagramas de secuencia

6.2.1.1 Aplicación Web Tradicional.

En la figura 44 (a) se ve de color rojo los mensajes y objetos que se eliminan en la adaptación, con líneas diagonales los objetos que son modificados.

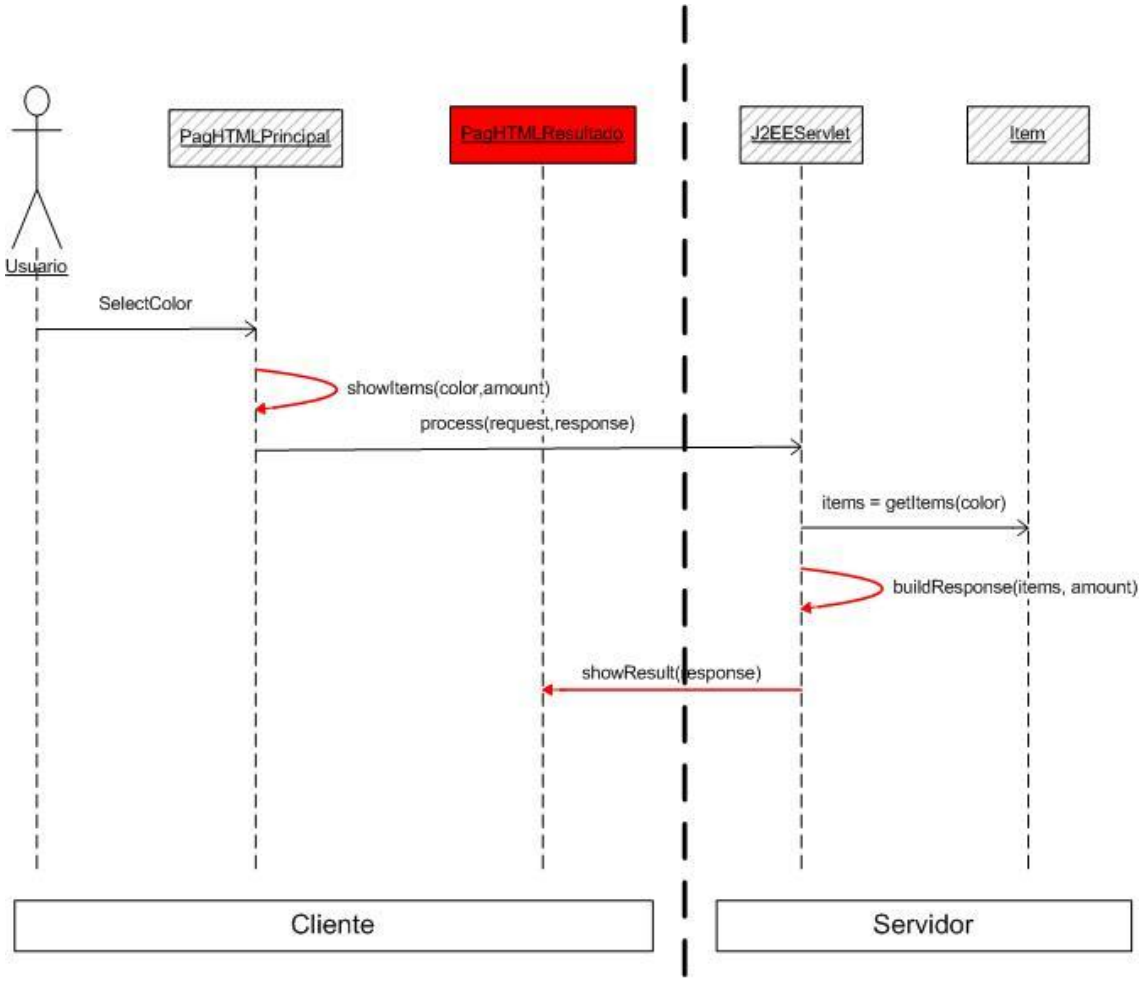


Figura 44 (a): Diagrama de secuencia de nuestra aplicación web tradicional de ejemplo.

6.2.1.2 Aplicación Web RIA

En la figura 44 (b) se ve de color azul los mensajes y objetos que se agregan en la adaptación, con líneas diagonales los objetos que son modificados.

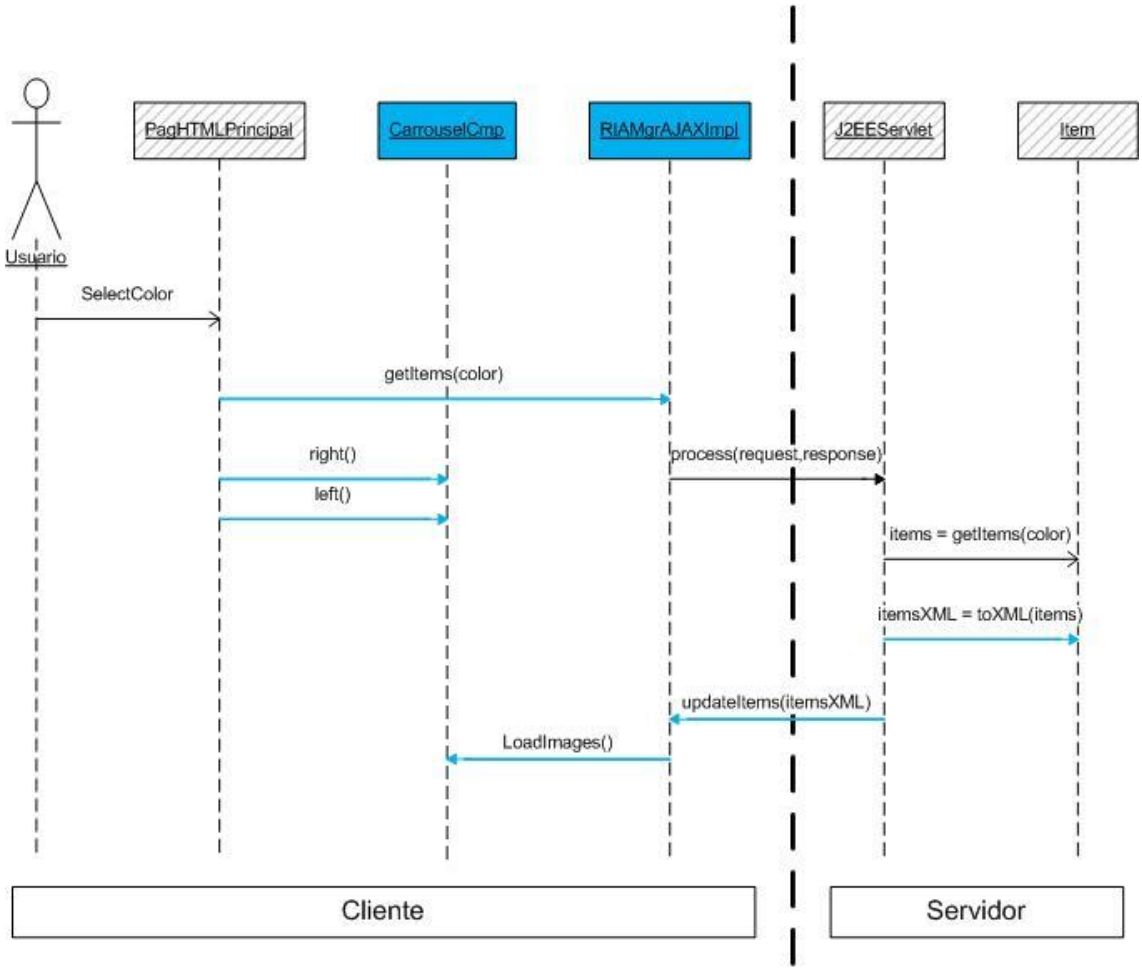


Figura 44 (b): Diagrama de secuencia de nuestra aplicación web RIA de ejemplo.

6.3 Costo de la adaptación

Sobre los dos puntos principales de la adaptación descritos arriba, se elaboraron las siguientes conclusiones.

1 Para una aplicación Web tradicional es necesario el desarrollo del componente visual en alguna tecnología que permita ser incluido en nuestra página HTML. Se debe incluir una llamada a la inicialización del mismo. Para la comunicación asincrónica entre el componente visual y la capa en el servidor existen diversos frameworks y librerías que evitan tener que reescribir código y aceleran el proceso de adaptación.

2 Sobre una aplicación Web tradicional con una arquitectura orientada a objetos; donde las capas dialogan entre ellas con mensajes que envían y devuelven objetos, la adaptación requerida es una traducción de los objetos a la representación XML acordada con la capa RIA.

En caso de encontrarse con una aplicación cuyas capas no están separadas funcionalmente y por ejemplo, el servlet accede a consultar los datos a una base de datos o los recupera mediante un protocolo determinado desde una aplicación legacy, de igual manera, se presenta la necesidad de construir una respuesta en el formato XML acordado. La diferencia radica en que dicho proceso puede ser engorroso y requerir diferentes tipos de modificaciones, elevando la complejidad de las tareas.

6.4 Conclusiones generales de la adaptación.

Resumiendo las tareas necesarias para la adaptación:

1. Diseño gráfico para el componente visual.
2. Utilización de frameworks, librerías o programación javascript con uso de XMLHTTPRequest.
3. Modificación en el servidor en el lenguaje requerido (java, perl, etc.)

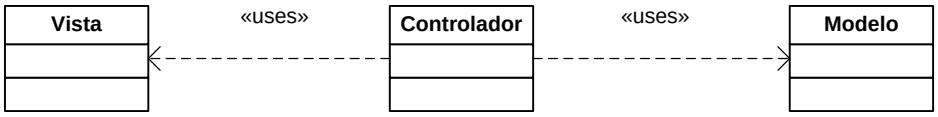
Hay una clara separación de tecnologías y conocimientos, lo cual requiere diversos perfiles pero permite una asignación bien diferenciada de tareas y responsabilidades.

Capítulo 7. Evaluación de la transformación de una aplicación Web tradicional a RIA utilizando patrones de diseño

7.1 Uso de patrones conocidos para desacoplar las capas de la aplicación

Los patrones de diseño ayudan a mantener una aplicación Web extensible, flexible y facilitar su mantenimiento [5].

Previamente se mostró uno de los patrones más utilizados para separar el cliente, el control del flujo de la aplicación y el modelo de la aplicación: MVC.



Gracias a esta separación no se debe crear una nueva aplicación para cada cliente que accede a la misma, duplicando el código que accede a los datos, extendiendo el tiempo de desarrollo innecesariamente.

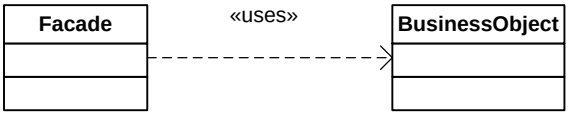
En este diseño, la capa de la vista puede acceder a los datos del modelo y decidir como presentar los mismos al cliente.

Otro problema comúnmente presentado es el acceso al modelo de la aplicación. Determinadas peticiones del cliente requieren interacciones con diferentes objetos del modelo.

En el ejemplo anterior no puede verse este punto, ya que la única información requerida se obtiene invocando a un método del objeto “item”.

Para aplicaciones reales es conveniente y sugerido el uso de un patrón de diseño que agrupe dichas interacciones en un solo servicio de aplicación.

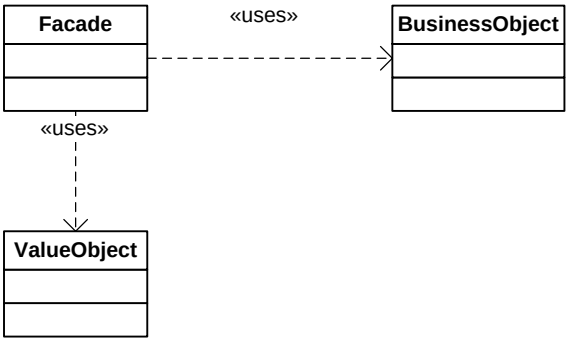
Un patrón que da solución a este problema es el Facade.



El patrón Facade provee una interfaz unificada de los servicios provistos por el modelo de la aplicación. Cualquier cambio en los servicios de la aplicación solo resulta en una modificación al “Facade”.

No hay necesidad de modificar la capa del controlador porque continúan utilizando la misma interfaz.

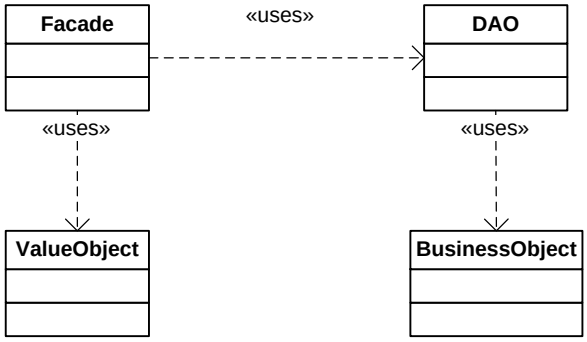
El cliente del modelo no siempre necesita actuar en forma transaccional modificando los datos del modelo ni accede a todas las propiedades de los objetos. Por este motivo, no es necesario que el cliente acceda a los objetos del negocio directamente, sino a través de vistas de los mismos que sirven tanto para restringir el acceso en forma directa y también para mejorar la performance de comunicación entre las capas. Patrones que dan esta solución son: VO, DTO,…



Otra capa que es necesario desacoplar es el acceso a las fuentes de datos de la aplicación. Para esto es recomendable un patrón como el DAO (Data Access Object).

Por ejemplo, si se cambia la base de datos de la aplicación y se requiere modificar las sentencias para acceder a ella, solo es necesario modificar la clase que accede al recurso de datos.

Las sentencias necesarias para esto no están distribuidas por todo el modelo, solo en los “DAO” correspondientes.



Como resultado se puede resumir que aplicaciones que usan patrones de diseño como ser Facade o DAO para el acceso a la lógica y datos del negocio, facilitan y ordenan la transformación.

No es parte del alcance de este trabajo ahondar en los patrones expuestos, para mayor información es posible consultar las referencias.

7.2 Cambio en la aplicación de ejemplo haciendo uso de patrones

La aplicación ha sido modificada en la capa del servidor para mostrar un diseño aplicando patrones [13], [20].

La clase del modelo *Item* presenta la siguiente estructura:

```
public class Item {
   private String image;
   private String information;
   private String name;
   private float  price;…
/*Accessors */
}
```

Dicha clase es recuperada del recurso de datos mediante una clase especializada en dicha tarea: ItemDAOImpl.

Para este ejemplo esto sigue siendo un proceso estático.

```
public class ItemDAOImpl {
   public static ItemDAOImpl INSTANCE = new ItemDAOImpl();
   public static List getBlueItems(){
      List list = new ArrayList();
```

```
        for (int i=0;i<15;i++){
            Item item = new Item();
            item.setImage("Imagenes/IB" + (i + 1) + ".jpg");
            item.setInformation("El Item Azul: " + i + " esta disponible.");
            item.setName("Item número " + i);
            item.setPrice((float)(i * 0.013));
            list.add(item);
        }
        return list;
    }
    public static List getRedItems(){
        List list = new ArrayList();
        for (int i=0;i<15;i++){
            Item item = new Item();
            item.setImage("Imagenes/IR" + (i + 1) + ".jpg");
            item.setInformation("El Item Rojo: " + i + " esta disponible.");
            item.setName("Item número " + i);
            item.setPrice((float)(i * 0.011));
            list.add(item);
        }
        return list;
    }
}
```

La lógica de negocio es accedida utilizando el patrón Facade.

```
public class ItemFacadeImpl {
    public static ItemFacadeImpl INSTANCE = new ItemFacadeImpl();
    public List getItems(String itemType){
        List items = new ArrayList();
        if (itemType != null) {
            if ("blue".equals(itemType)) {
                items = ItemDAOImpl.INSTANCE.getBlueItems();
            } else if ("red".equals(itemType)) {
                items = ItemDAOImpl.INSTANCE.getRedItems();
            }
        }
        return ItemVOAssembler.toVOs(items);
    }
}
```

Como se puede ver el Facade no devuelve una lista de ítems directamente, sino lo que devuelve el método *toVOs* de la clase *ItemVOAssembler*. Esta clase es la responsable de ensamblar un objeto VO a partir de un objeto de negocio.

El objeto VO es una copia del objeto de negocio que solo muestra los atributos que el cliente requiere en este escenario.

```
public class ItemVOAssembler {
    public static List toVOs(List list){
        Iterator iterator = list.iterator();
        List result = new ArrayList();
        while (iterator.hasNext()){
            Item item = (Item) iterator.next();
            ItemVO itemVO = new ItemVO();
            itemVO.setImage(item.getImage());
            itemVO.setInformation(item.getInformation());
            result.add(itemVO);
        }
        return result;
    }
}
public class ItemVO {
    private String image;
    private String information;
    public String getImage() {
        return image;
    }
    public String getInformation() {
        return information;
    }
}
```

A continuación el “controlador” (servlet) para la aplicación Web tradicional:

```
public class CarrouselControllerTradicional extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String itemType = request.getParameter("itemType");
            int itemsAmount = Integer.parseInt(request.getParameter("itemsAmount"));
            List items = ItemFacadeImpl.INSTANCE.getItems(itemType);
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Carrousel Tradicional</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<table width=\"761\" border=\"1\" height=\"131\">");
            out.println("<tr height=\"83\"><td><table width=\"747\" border=\"0\" height=\"124\">");
            out.println("<tr height=\"94\"><td><table height=\"92\" width=\"693\" border=\"0\"><tr height=\"84\">");
```

```
            for (int i = 0; i<itemsAmount;i++){
                out.println("<td width=\"74\"><img src=\"" +  ((ItemVO) items.get(i)).getImage() + "\" height=\"85\" width=\"85\"
align=\"middle\" border=\"0\"></td>");
            }
            out.println("</tr></table></td></tr></table></td></tr></table>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

7.2.1 El nuevo diagrama de secuencia

En color rojo los mensajes y objetos que se eliminan en la adaptación, con líneas diagonales los objetos que son modificados, como podemos ver en la figura 45 (a).

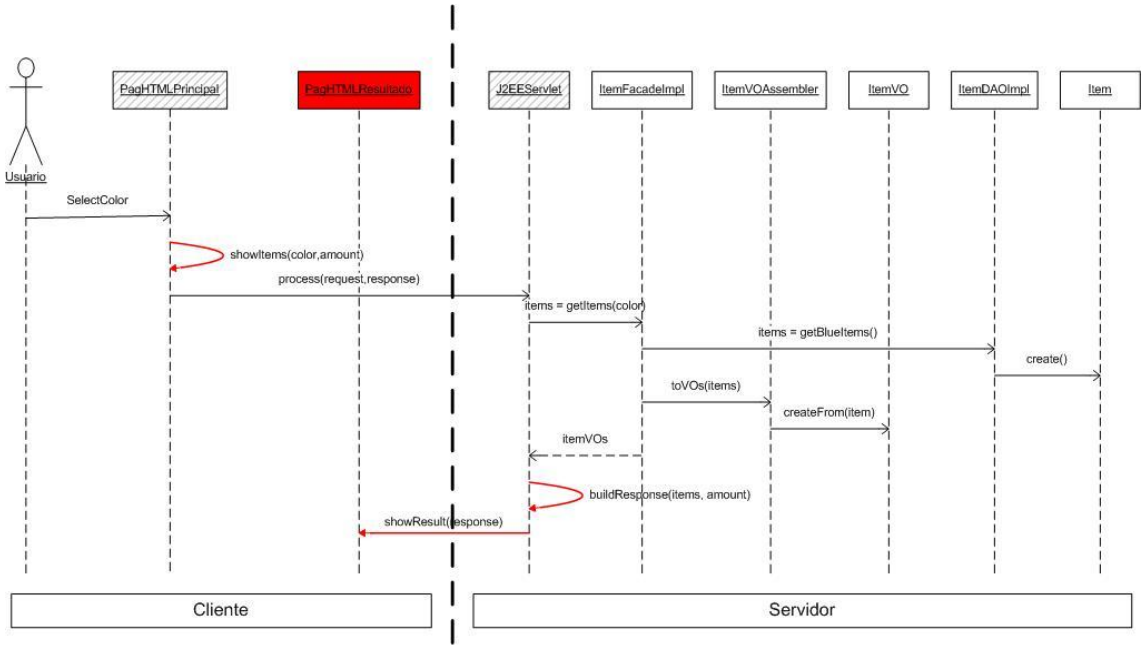


Figura 45 (a): Diagrama de secuencia de nuestra aplicación web tradicional de ejemplo.

7.2.2 El nuevo Controlador

El nuevo diseño muestra claramente que el cambio necesario en la adaptación del lado del servidor ocurre en la capa del controlador.

Se necesita crear un nuevo controlador que se encargue de obtener la nueva representación de los ítems esperados y los devuelva a la capa del cliente que lo solicitó.

Para obtener la nueva representación, se puede agregar esta funcionalidad en la clase ItemXMLAssembler, la cual es responsable de la transformación del *Value Object* en su representación XML esperada por el cliente:

```
public class ItemXMLAssembler {
   public static String toXML(List list){
      Iterator iterator = list.iterator();
      String xml = "<?xml version=\"1.0\"?>" +
            "<items>";
      while (iterator.hasNext()){
         Item item = (Item) iterator.next();
         xml = xml + "<item><image>" + item.getImage() + "</image><information>" + item.getInformation()+
"</information></item>";
      }
      xml = xml + "</items>";
      System.out.println(xml);
      return xml;
   }
}
```

Y finalmente el nuevo controlador queda modificado de la siguiente manera:

```
public class CarrouselControllerRIA extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         String itemType = request.getParameter("itemType");
         List items = ItemFacadeImpl.INSTANCE.getItems(itemType);
         String itemsXml = ItemXMLAssembler.toXML(items);
         // Write XML to response.
         response.setContentType("application/xml");
         response.getWriter().write(itemsXml);
      } finally {
         out.close();
      }
```

```
    }
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

7.2.3 El nuevo diagrama de secuencia II

La figura 45 (b) muestra el nuevo diagrama de secuencia, donde se ve en color azul los mensajes y objetos que se agregan en la adaptación, con líneas diagonales los objetos que son modificados (o creadas nuevas versiones).

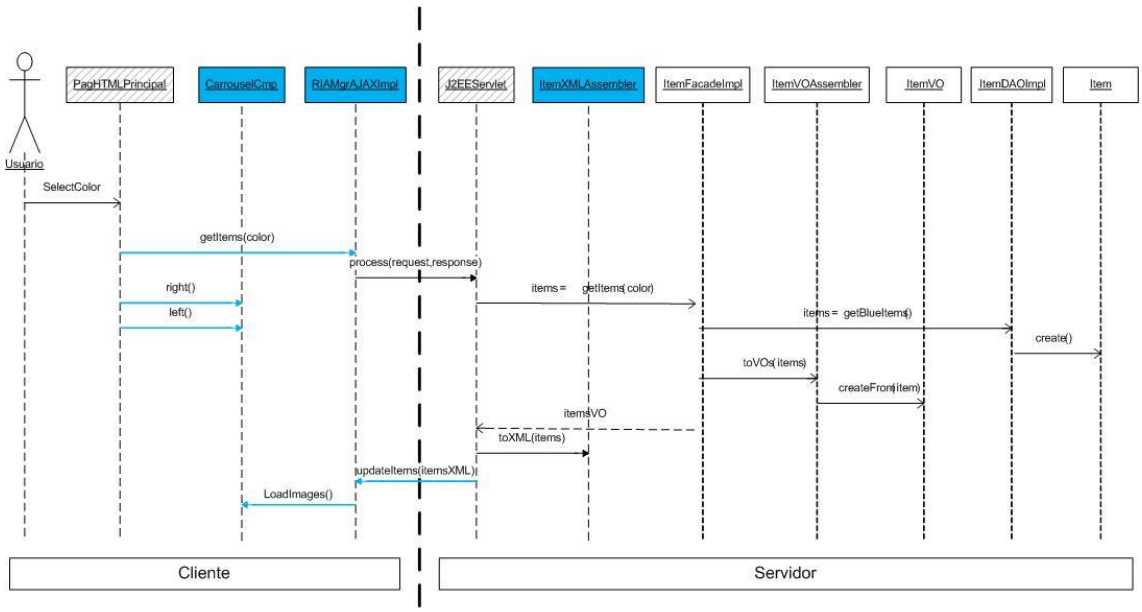


Figura 45 (b): Diagrama de secuencia de nuestra aplicación web RIA de ejemplo.

7.4 Modificación en las Pruebas de la aplicación.

El cambio a RIA en una aplicación lleva a la modificación de las pruebas realizadas en la capa del cliente.

En una aplicación típica, muchas aéreas requieren ser probadas.

Comenzando por los componentes más simples, clases, los desarrolladores necesitan programar pruebas unitarias (unit tests) para asegurar que las partes

más pequeñas de la aplicación funcionan correctamente. Cada componente puede superar la prueba solo, sin embargo, los desarrolladores necesitan asegurar que funcionan juntos de la manera esperada, como parte de un subsistema, como parte de todo el sistema; entonces las pruebas de *integración* deben realizarse. Otros tipos de pruebas son realizadas en determinados proyectos según se requiera:

- Unitario – verificación de clases individuales.
- Módulo - prueba de interacción de grupos de clases (paquetes).
- Integración – verificación de interacciones entre módulos/componentes.
- Funcional – verificación del comportamiento externo de los módulos, componentes and sistemas.
- Sistema – prueba del sistema sobre los objetivos.
- Aceptación - validación de la aplicación sobre los requerimientos del usuario (casos de uso).
- Regresión – re-ejecutar todas las pruebas del sistema cuando este es modificado.

Todas estas pruebas se realizan generalmente con ayuda de frameworks especializados.

No es el propósito de este trabajo ahondar en las metodologías y herramientas disponibles para realizar pruebas como ser: Selenium, Squish, GWTTestCase, JWebUnit [33], etc. sino mostrar los cambios en las pruebas y puntos a tener en cuenta cuando se crea una aplicación RIA a partir de una aplicación existente.

Para esto se toman dos frameworks disponibles del mercado (HTTPUnit y JSUnit) para mostrar un ejemplo.

HTTPUnit es un framework basado en JUnit [36] el cual permite la ejecución automática de Tests para las aplicaciones Web.

Como se va a probar una aplicación Web, la herramienta debería comportarse de igual manera que los navegadores que usan los usuarios. La aplicación no debe hacer diferencias de los servicios invocados desde un navegador o una herramienta de pruebas.

HTTPUnit simula exactamente las peticiones GET y POST de un navegador normal y provee un modelo de objetos con el cual codificar las pruebas.

7.5 Caso de prueba para el componente Carrusel en la aplicación Web Tradicional

El caso de prueba realizado será para el comportamiento del componente Carrousel.

Para esto se utiliza solo un pequeño conjunto de clases de HTTPUnit que se usan habitualmente. Una sesión de usuario (una secuencia de iteraciones con la aplicación Web) es encapsulada en una "WebConversation". El framework entonces retorna un "WebResponse", conteniendo la página resultado y los atributos del servidor.

Teniendo en cuenta ambos HTMLs de la aplicación Web tradicional (CarrouselTradicional y CarrouselServletTradicional) un simple escenario de prueba podría ser como se muestra en el código de abajo.

```
/**
* Verifica que enviando el formulario de selección del usuario
* con el valor "blue" resulta en la página conteniendo al menos una imagen
* "jpg" del componente Carrousel.
**/
public void testCarrouselTradicional () throws Exception {
    WebConversation conversation = new WebConversation();
    WebRequest request = new GetMethodWebRequest(
"http://localhost:8080/CarrouselExample/CarrouselTradicional.html" );
    WebResponse response = conversation.getResponse( request );
    WebForm loginForm = response.getForms()[0];
    request = loginForm.getRequest();
    request.setParameter( "itemType", "blue" );
    response = conversation.getResponse( request );
    assertTrue( "Carrousel not filled",
        response.getText().indexOf( ".jpg" ) != -1 );
}
```

7.6 Caso de prueba para el componente Carrusel en la aplicación RIA

Para probar la aplicación RIA en la capa del cliente, es necesario modificar el escenario de prueba de manera de poder acceder al componente Carrousel que ahora se encuentra encapsulado en javascript.

Existen diversos frameworks para pruebas de aplicaciones RIA (algunos mencionados anteriormente), los mismos deben proveer solución a un tema importante en las pruebas para RIA: la comunicación asincrónica entre el cliente y el servidor.

Esto requiere que el escenario de prueba contemple una espera de tiempo para verificar resultados.

El caso de prueba es muy simple y solo se evaluará la carga del componente como los hicimos en el escenario de prueba anterior.

Para esto se usará JsUnit que es un framework para pruebas en javascript.

Las pruebas unitarias en JsUnit se llaman Funciones de pruebas. Las funciones de pruebas viven en una página HTML llamada Página de pruebas. Una Página de pruebas es cualquier HTML que incluye (tiene un “include”) el script: jsUnitCore.js. Este script provee funciones de afirmación como por ejemplo: assertEquals.

Teniendo en cuenta el nuevo HTML y el código javascript de la aplicación Web RIA un simple escenario de prueba podría ser como se muestra en el código de abajo.

Se crea un nuevo documento HTML con el código HTML donde es mostrado el Carrousel. El código javascript que permite girar a derecha e izquierda y cargar inicialmente las imágenes, fue encapsulado en un nuevo script que se llama: aplicacionRIA.js.

La función testGetItems es interpretada por el JsUnit TestRunner, primero obtiene los ítems para el color “blue”, luego de 1 segundo de espera hace un corrimiento a derecha. Guarda la imagen que se encuentra en la posición dos. A continuación realiza un corrimiento a izquierda y verifica que la imagen de la posición uno sea la que anteriormente se encontraba en la posición dos.

```
<html>
<head>
<script type="text/javascript" language="javascript" src="aplicacionRIA.js"></script>
<script language="javascript" src="/path/to/jsunit/app/jsUnitCore.js"></script>
<script language="javascript">
function testGetItems(){
        var image i;
        getItems('blue');
        //después de 1 segundo gira a la derecha
```

```
        setTimeout(“right()”,1000);
        i = document.images[2].src;
        left();
        assertTrue("Luego de girar a izquierda la imagen de la posición 2 debería estar en la posición 1",
        document.images[1].src == i);
}
</script>
</head>
<body>
<table width="761" border="1" height="131">
        <tr height="83"><!-- Row 1 -->
        <td>
        <table width="747" border="0" height="124">
                <tr height="94"><!-- Row 1 -->
                <td>
                <table height="92" width="693" border="0">
                <tr height="84">
                 <td width="74"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="70"><img height="85" alt="" width="85" align="middle" border="0"></td>
                 <td width="74"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="69"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="73"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="81"><img height="85" alt="" width="85" align="middle" border="0"></td>
                <td width="62"><img height="85" alt="" width="85" align="middle" border="0"></td>
                        <td><img height="85" alt="" width="85" align="middle" border="0"></td>
                </tr>
                </table>
                </td>
                </tr>
        </table>
        <img align="right" height="18" alt="" src="Imagenes/right.png" width="18" border="0"
        onclick="right();">
        <img align="right" height="18" alt="" src="Imagenes/left.png" width="18" border="0"
        onclick="left()">
        </td>
        </tr>
</table>
</body>
</html>
```

7.7 Tests utilizando grabadores de Macros.

Los grabadores de macros son programas que permiten grabar la secuencia de pasos realizada por un usuario en una aplicación y luego repetirla.

Existen programas para grabar macros en navegadores Web como es el caso de iMacro.

Una de las prestaciones de estos programas, es la posibilidad de utilizarlos para realizar tests de aplicaciones Web.

iMacros además permite grabar una secuencia de navegación de un sitio y verificar la existencia de diferentes componentes en él.

Para el caso de las aplicaciones RIA, iMacros puede automatizar pruebas usando comandos basados en HTML TAGs estándares. Y puede automatizar absolutamente todos los elementos AJAX con la tecnología DirectScreen, simulando los “clicks” del mouse dentro de la pantalla del navegador.

Para el ejemplo del Carrousel se grabó una secuencia de navegación básica utilizando el navegador de iMacros (versión prueba).

La figura 46 muestra el navegador con el plug-in de iMacros instalado, a la espera de ejecutar la secuencia de prueba.

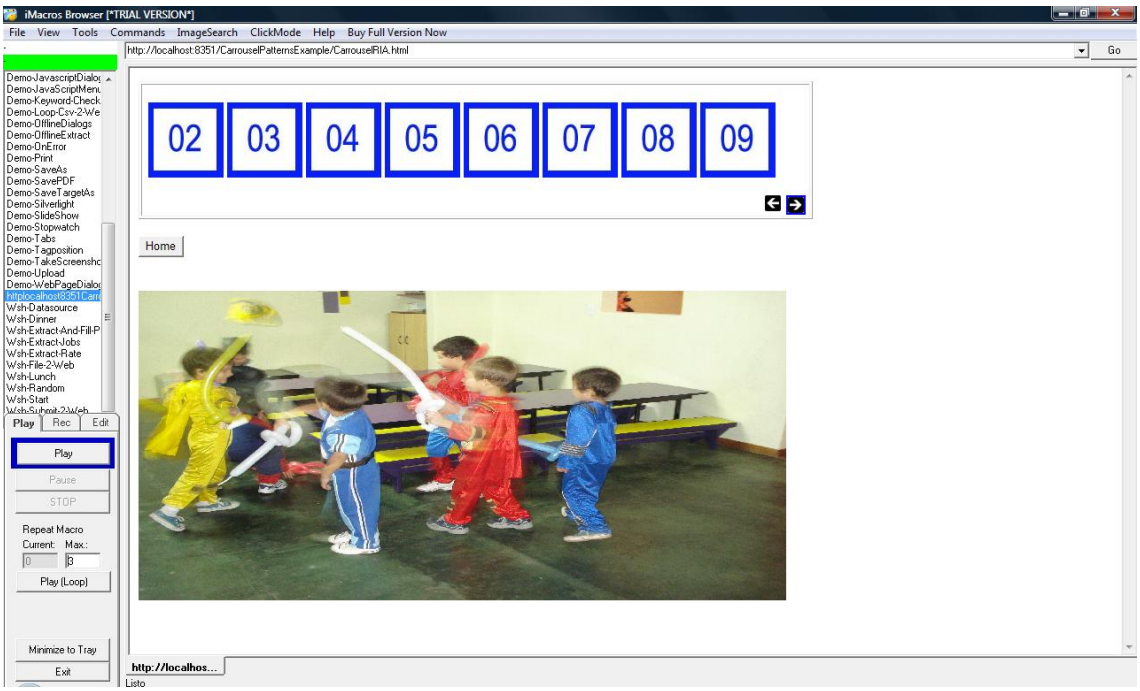


Figura 46: Navegador con plug-in de iMacros.

Se visualizan o editan los comandos de la secuencia:

```
VERSION BUILD=6200707
TAB T=1
TAB CLOSEALLOTHERS
URL GOTO=http://localhost:8351/CarrouselPatternsExample/CarrouselRIA.html
TAG POS=1 TYPE=INPUT:BUTTON FORM=NAME:NoFormName ATTR=VALUE:Ingresar
TAG POS=1 TYPE=IMG ATTR=HREF:http://localhost:8351/CarrouselPatternsExample/Imagenes/right.png
```

Con la sentencia URL GOTO vamos a la página de inicio de nuestro ejemplo.
Luego se presiona el botón “Ingresar”.
Finalmente se aprieta el botón de navegación del Carrousel a la derecha.
La imagen de arriba muestra la aplicación luego de que la macro fue ejecutada.
Se ve que luego de girar a la derecha como último paso de la macro, la primera imagen visible en el Carrousel es la segunda (la número dos).
Se decide verificar que una de las imágenes del Carrousel se encuentre cargada. Para esto, como la carga de la imagen es dinámica y no aparece en el TAG IMAGE de la página HTML es necesario hacer uso de otra forma de verificar que la imagen existe en la página. iMacro provee otra herramienta para buscar imágenes dentro de una página: el PlugIn de reconocimiento de imágenes.
A diferencia de los comandos de DirectScreen, los comandos IMAGESEARCH e IMAGECLICK no se basan en las coordenadas de un elemento sino en su apariencia visual.
Para esto se selecciona una parte visual de la pantalla y se la guarda como un BMP de 24-bit con el nombre: *Image2.bmp*.
Esta parte guardada que se quiere comparar, se la puede buscar en la página incluyendo el comando IMAGESEARCH en la macro.

```
VERSION BUILD=6200707
TAB T=1
TAB CLOSEALLOTHERS
URL GOTO=http://localhost:8351/CarrouselPatternsExample/CarrouselRIA.html
TAG POS=1 TYPE=INPUT:BUTTON FORM=NAME:NoFormName ATTR=VALUE:Ingresar
TAG POS=1 TYPE=IMG ATTR=HREF:http://localhost:8351/CarrouselPatternsExample/Imagenes/right.png
IMAGESEARCH IMAGE=Image2.bmp CONFIDENCE=95
```

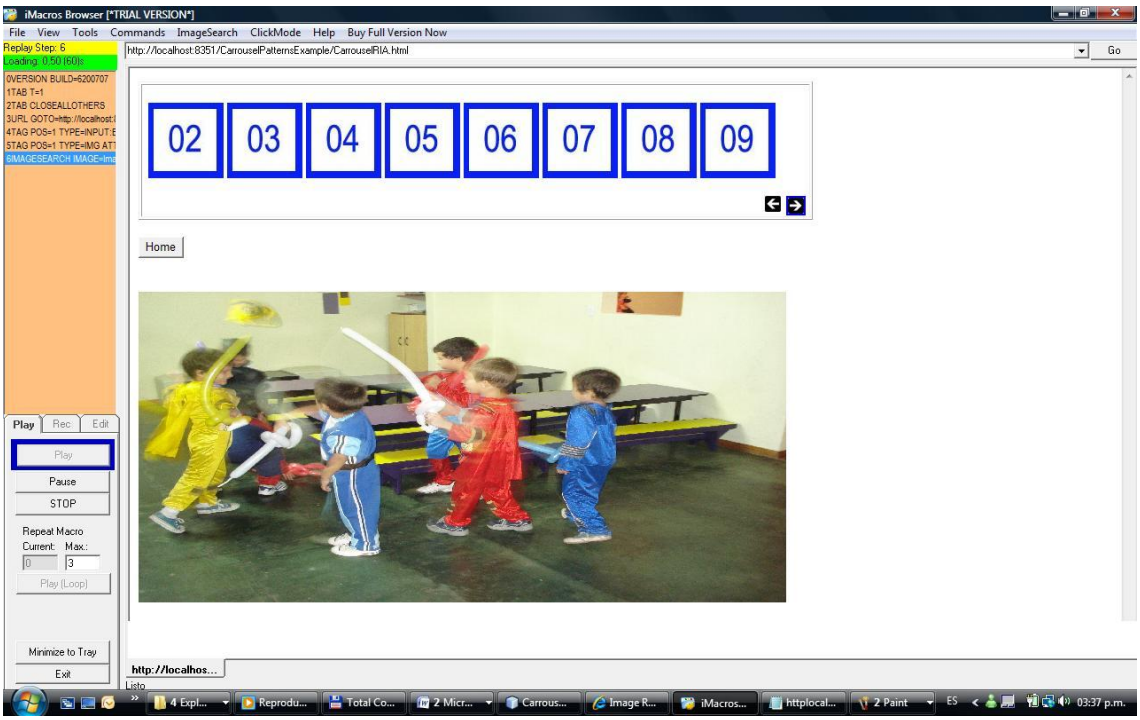


Figura 47: Ejemplo de ejecución de secuencia de iMacros.

La figura 47 muestra en la parte de arriba a la izquierda del navegador, los pasos de la macro y con color verde muestra que la última sentencia de buscar la imagen fue exitosa.

Ahora, si se comenta la parte de código que retorna las imágenes para el color Azul, como se puede ver en la figura 48:

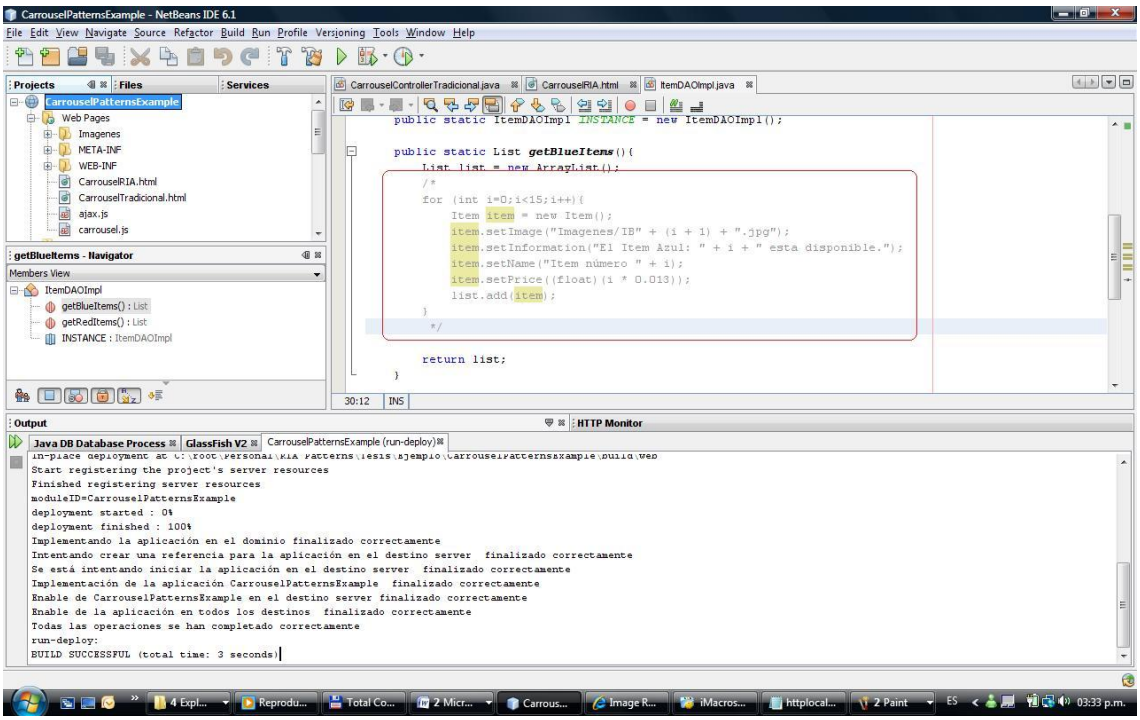


Figura 48: Modificación de código en el editor NetBeans.

Cuando se corre nuevamente la macro se encuentra un error como se ve en esquina superior izquierda de la figura 49.

Figura 49: iMacros mostrando un error en la ejecución del test.

7.8 Utilización de ADV-Charts en la realización de Tests

Como ha sido tratado en capítulos anteriores, uno de los objetivos principales de este trabajo es mostrar la inclusión de nuevos diagramas en la especificación de Patrones de Diseño Web con el fin de completar y mejorar las mismas.

En la realización de pruebas al nuevo componente RIA incluido en la aplicación de ejemplo, se debe hacer un seguimiento de los diferentes comportamientos del mismo ante las interacciones del usuario.

Justamente estos comportamientos ante eventos del usuario están definidos en el ADV-Chart incluido en la especificación.

Lo que se plantea en este apartado es la posibilidad y utilidad de guiar las pruebas en base a los eventos definidos en el ADV-Chart.

Así, los pasos probados anteriormente coinciden justamente con los primeros dos eventos detallados en el ADV-Chart del componente Carrousel, marcados en los color rojo en la figura 50.

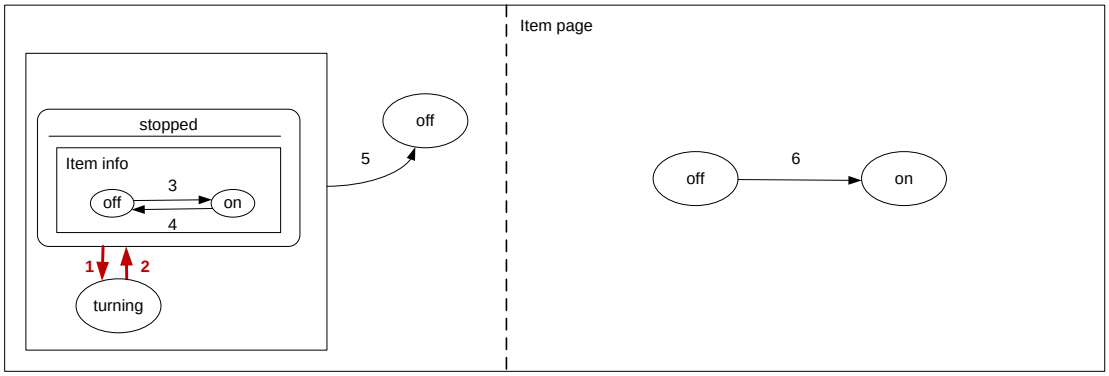


Figura 50: ADV-Chart que resalta los eventos probados en el Test.

1:

*Event: TurnLeftClick / **TurnRightClick***

*Pre-cond: **Focus(** turnLeft/**turnRight** **)***

*Post-cond: carrusel.moveLeft() / **carrusel.moveRight()**;*

Observación: primer paso de la prueba realizada (desplazar a la derecha los elementos del Carrousel).

2:

Event: MouseUp

Pre-cond: ***Focus(*** *turnLeft/****turnRight)****;* ***carrousel.isTurning==true***

Post-cond: carrousel.isTurning==false; perCont = perCont – ***items(****left/****right); perCont = perCont + items(****right/****left);***

Observación: se modifica el contenido del componente incluyendo un nuevo elemento a la izquierda y quitando uno a la derecha (verificación de la imagen número dos en la primera posición de la izquierda).

7.9 Conclusiones generales del cambio en los Tests de la adaptación

Como se mostró en los ejemplos anteriores los Test de la aplicación deben ampliarse para poder verificar el correcto comportamiento de las nuevas funcionalidades RIA.

Básicamente, el objetivo de la modificación de la aplicación será interactuar con el usuario de una manera más eficiente y amigable. Por ello, la funcionalidad nueva deberá probarse en la capa del cliente, donde el usuario espera el resultado.

Existen diferentes herramientas para ayudar a realizar pruebas unitarias e integrales en aplicaciones RIA. Como se vio en los ejemplos de arriba, iMacro y jsUnit son dos posibilidades.

Capítulo 8. Trabajos Relacionados

8.1 Introducción

En los capítulos anteriores se trataron diferentes temas relacionados con las etapas del proceso de transformación de una aplicación web tradicional a RIA.
Es importante poder revisar en este capítulo los trabajos más influyentes que fueron de fundamental aporte para lograr el objetivo de esta tesis y las conclusiones respecto a la bibliografía encontrada y disponible sobre el tema.

8.2 Aplicaciones RIA

Existen muchos trabajos que dan una introducción a las aplicaciones RIA, mediante diferentes enfoques.
Por ejemplo en [11] se define a una aplicación RIA mencionando sus ventajas y beneficios respecto a las aplicaciones Web tradicionales.
Como valor agregado a otros trabajos similares, se mencionan algunos de los potenciales riesgos en la experiencia del usuario con RIA.
Si bien las aplicaciones “ricas” pueden mejorar en mucho la experiencia del usuario, existen algunos riesgos. Como con toda otra técnica existe siempre la tentación de sobre usarla y abusar de ella. Las interfaces ricas deben ser usadas donde agregan valor a la experiencia del usuario, no para sumar campanas y sonidos o mostrar que el sitio es “cool”.
Conocimiento sobre los usuarios y sus tareas ayudarán a definir donde es mejor utilizar la funcionalidad rica.
Entre los pasos previos que un consultor debe llevar a cabo en el proceso de transformación de una aplicación Web tradicional a RIA, se pueden mencionar los siguientes:

- Determinar cuando la transformación mejorará la experiencia del usuario.
- Conocimiento de los usuarios.
- Los controles deben comunicar su propósito y función.
- Notificar al usuario cuando la respuesta no es inmediata.

- Considerar soluciones para usuarios que no pueden usar RIA.

8.3 Especificación

Existen diversos trabajos realizados sobre como especificar las nuevas funcionalidades de las aplicaciones RIA, sus comportamientos e interacciones con el usuario, mediante lenguajes y métodos apropiados, uso de herramientas nuevas y adaptadas, etc. [2], [26], [29] y [31].

Para este trabajo se decidió optar por utilizar el método OOHDM por varios motivos, principalmente el conocimiento previo sobre dicha metodología, nuestra cercanía con uno de los autores y personas que investigan diferentes aplicaciones del método, la diversidad de material y fundamentalmente los trabajos realizados para extender su alcance a las aplicaciones RIA.

Entre otros materiales consultados es importante resaltar la extensión de *Web Models* para dar soporte a las aplicaciones RIA mediante un nuevo método llamado RUX-Model [21].

La bibliografía consultada de OOHDM es extensa, como introducción a la metodología fue necesario consultar entre otras fuentes a [28].

Es de gran valor para la especificación de aplicaciones RIA los trabajos realizados utilizando ADV Charts [34], definiendo aspectos teóricos y analizando patrones web en aplicaciones RIA actuales.

Dichos trabajos fueron parte del origen del objetivo central de esta tesis, cuyo propósito principal es la mejora de las especificaciones actuales para patrones Web, es decir la especificación de comportamiento RIA.

Gran parte de los patrones mostrados en los ejemplos se encuentran en varios sitios que describen patrones Web, mostrando galerías y catálogos de los mismos, por ej.: [32], [40], [39] y [37].

8.4 Migrando aplicaciones tradicionales a RIA

Actualmente, en su mayoría, existe dentro de la bibliografía RIA gran cantidad de bibliografía especializada en tecnologías RIA, por ejemplo AJAX, Flex,

OpenLaszlo, etc., pero es poca la bibliografía sobre el proceso de transformar o migrar una aplicación tradicional a RIA.

Es importante remarcar la necesidad de tener un conocimiento general de las tecnologías a la hora de transformar una aplicación tradicional a RIA.

Para todas las tareas (especificación, diseño, arquitectura, tests, etc.) involucradas en el desarrollo de una aplicación Web tradicional se debería realizar un análisis para determinar los posibles cambios durante la migración a RIA. Esto fue uno de los objetivos de este trabajo.

Existen autores [42] que advierten y sugieren realizar una transformación gradual, evitando la necesidad de reemplazar en su totalidad sistemas que funcionan eficientemente.

El equipo por otro lado, puede no estar listo para descartar todas las viejas tecnologías y herramientas para mover a las nuevas opciones.

Como se mostró en los capítulos 6 y 7 es una gran ventaja al momento de transformar una aplicación tradicional, partir de una aplicación correctamente diseñada. Teniendo una clara separación de capas y responsabilidades, permite introducir la nueva lógica en forma ordenada.

Existe mucha bibliografía respecto al uso de patrones de diseño y el uso de los mismos en aplicaciones Web tradicionales.

Conclusiones y trabajo futuro

Cumpliendo con el objetivo de este trabajo, se ha explicado que son las aplicaciones RIA, su diferencia con las aplicaciones web tradicionales, ventajas, desventajas y posibles riesgos en el uso de la tecnología RIA.

Se ha dado una visión introductoria a diferentes tecnologías RIA, con ejemplos reales del uso de AJAX para implementar un patrón de diseño Web.

Se mostraron los cambios realizados para transformar una aplicación tradicional a RIA, el cambio en su arquitectura y como cambia la transformación entre una aplicación web tradicional correctamente diseñada y otra que no tiene un buen diseño de objetos y una correcta separación de capas.

Una sección muestra como realizar tests en aplicaciones RIA, dando ejemplos con una de las herramientas disponibles en el mercado.

Durante este trabajo pudo verse que planificar el éxito de una aplicación RIA no es solo escribir buen código. Las implementaciones exitosas requieren un trabajo de consultoría previo a las consideraciones tecnológicas, principios de diseño, perfiles e infraestructura de la organización.

El aporte más importante de este trabajo es fundamentalmente la mejora propuesta para las especificaciones de patrones Web actuales mediante la inclusión de ADVs y ADVCharts en las mismas, agregando mayor detalle visual y formalizando la definición del comportamiento esperado.

De continuar la investigación y el trabajo en esta área, se encuentra de interés y utilidad poder definir una metodología y técnica para transcribir los ADVCharts en lenguajes de programación y tecnologías que permitan implementar el componente del patrón de diseño Web que hemos especificado.

Existen diversos métodos que incluyen herramientas y arquitecturas propuestas para implementar el diseño. Este trabajo escogió por diferentes motivos ya enunciados a OOHDM como metodología para diseño de aplicaciones Web.

Hay trabajos que desarrollan técnicas de implementación del diseño y mapean componentes OOHDM en una arquitectura J2EE.

Por ejemplo en [7] se plantea una posible arquitectura de aplicaciones Web y como mapearla con las capas y componentes de OOHDM.

El patrón MVC (explicado en este trabajo) si bien provee un conjunto de principios estructurales para construir aplicaciones iterativas de manera modular no cumple totalmente los requerimientos de las aplicaciones Web ya que se basa en una visión puramente transaccional del software. La mayor carencia del patrón MVC es que no tiene en consideración los aspectos de navegación que por argumentos ya mencionados, deberían ser apropiadamente soportados.

La presentación y la estructura de los datos son atendidas en el mismo componente de software, típicamente un objeto JSP. Así mismo la idea que la navegación ocurre siempre en un contexto y que la información relacionada al contexto debe ser provista al usuario está ausente en MVC.

De esta forma, si se busca que un objeto JSP sea sensible al contexto, se debe incluir parámetros en su llamada de manera de informarle los cambios de contexto y utilizar sentencias condicionales para tomar decisiones ante los cambios de contextos. Esto hace que el objeto JSP sea sobrecargado, difícil de mantener y evolucionar.

Resumiendo, el problema es que la lógica de navegación, los nodos y el manejo de contextos, es tratado dentro de los objetos JSP, mezclando la navegación con el diseño de la interface.

OOHDM-Java2 extiende la idea de MVC separando claramente los nodos de sus interfaces, así introduce la idea de objeto de navegación, también reconociendo que la navegación puede ser dependiente del contexto.

En la figura 51 se pueden ver los componentes de alto nivel de la arquitectura OOHDM-Java2, juntos con las iteraciones más importantes entre componentes cuando una petición es atendida. No todas las aplicaciones requieren utilizar todos los componentes de la arquitectura, aplicaciones más simples, como las aplicaciones de navegación de solo lectura, deben usar solo un subconjunto de estos componentes.

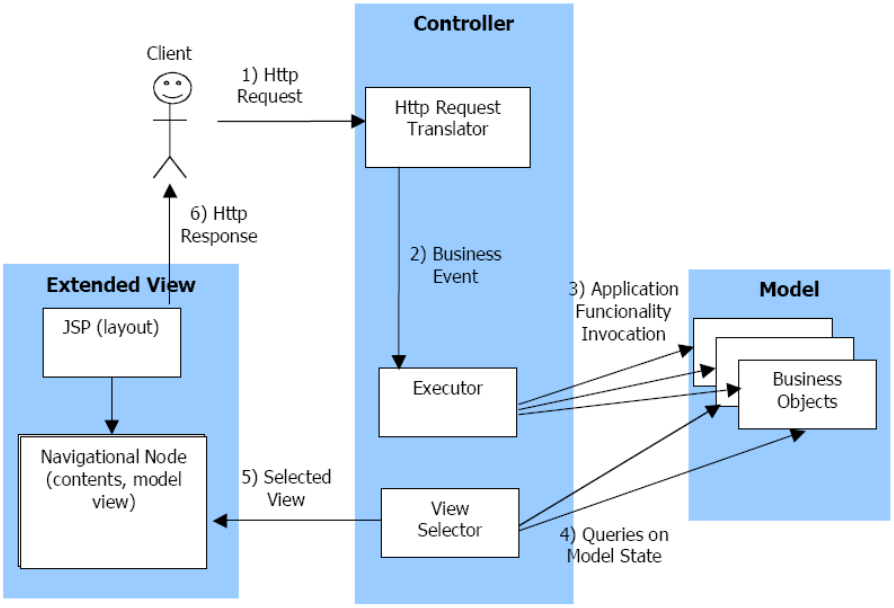


Figura 51: Componentes de la arquitectura OOHDM - J2EE [7].

Para las aplicaciones RIA se deberían agregar y modificar los componentes de la arquitectura para soportar acciones únicamente en el cliente y otras que solicitan información al Controlador pero sin refrescar o cambiar la página actual.

Esto fue descripto en el capítulo 6 y 7 de este trabajo, agregando el componente cliente a la aplicación, donde el cliente puede acceder mediante un link de navegación directamente al controlador, o interactuar con la capa cliente RIA y esta ser la encargada de comunicarse con el controlador cuando así lo requiera, sin modificar la página actual.

Un diagrama de la nueva arquitectura simplificada se puede ver en la figura 52.

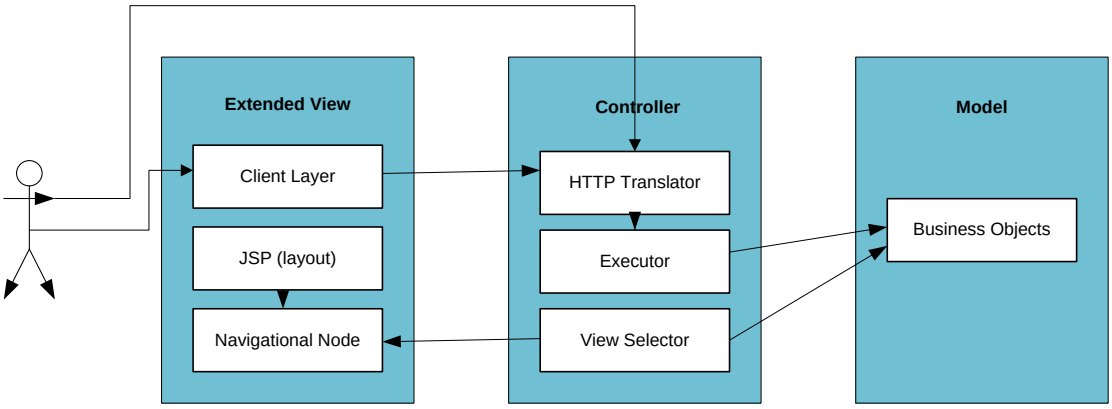


Figura 52: Componentes de la arquitectura OOHDM - J2EE modificada para incluir aplicaciones RIA.

Por último, para completar una investigación futura, se ve la necesidad de obtener el código que implementa el comportamiento definido en los ADV-Chart, para dicho autómata se podría partir de técnicas existentes que convierten StateCharts en código Java como puede verse en [17] o [24], ajustando las traducciones a otros lenguajes (por Ej.: javascript).

Bibliografía y Referencias

1. Anderson, David J. *Using MVC Pattern in Web Interactions.* junio 1, 2000. http://www.uidesign.net/Articles/Papers/UsingMVCPatterninWebInter.html (accessed noviembre 20, 2008).
2. Coffin, Carneiro, D. D. Cowan, C.J.P. Lucena, and D. Smith. *An Experience Using JASMINUM—Formalization AssistingWith the Design of User Interfaces.* London UK: Springer-Verlag, 1995.
3. Coremans, Chris. *AJAX and Flash Development with OpenLaszlo: A Tutorial.* Chicago: BRAINYSOFTWARE, 2006.
4. Crane, Dave, Eric Pascarello, and Darren James. *Ajax in Action.* London UK: Manning Publications, 2005.
5. Crupi, John, Deepak Alur, and Dan Malks. *Core J2EE Patterns: Best Practices and Design Strategies.* Upper Saddle River: Prentice Hall, 2003.
6. Deitel, Paul J., and Harvey M. Deitel. *AJAX, Rich Internet Applications, and Web Development for Programmers (Deitel Developer Series).* Upper Saddle River: Prentice Hall , 2008.
7. Douglas Jacyntho, Mark, Daniel Schwabe, and Gustavo Rossi. "A Software Architecture for Structuring Complex Web Applications." 2002.
8. E. Gamma, R. Helm, T. Johnson, J. Vlissides. *Design Patterns.* Old Tappan: Addison-Wesley Professional, 2003.
9. Eden, Amnon H., Gasparis Epameinondas, and Jonathan Nicholson. *Formal Pattern Design Specification with LePUS3 and Class-Z.* 1998. http://www.lepus.org.uk/ref/companion/Observer.xml (accessed Diciembre 1998, 16).
10. Fain, Yakov, Dr. Victor Rasputnis, and Anatole Tartakovsk. *Rich Internet Applications with Adobe Flex & Java (Secrets of the Masters).* New Jersey: SYS-CON Books, 2007.
11. Gallagher, R. "Improving User Experience with Rich Internet Applications." *www.google.com.* 2006. http://www.google.com (accessed Septiembre 2008, 13).
12. Garrett, Jesse James. "Adaptive Path." *www.adaptivepath.com.* 2005. http://adaptivepath.com/ideas/essays/archives/000385.php (accessed Septiembre 2008, 20).
13. Grand, Mark, and John Wiley. *Pattern in Java, vol. 1. Segunda edición.* Hoboken: JOHN WILEY & SONS LIMITED, 2002.
14. Hanson, Robert, and Adam Tacy. *GWT in Action: Easy Ajax with the Google Web Toolkit.* Greenwich: MANNING PUBLICATIONS COMPANY, 2007.
15. Harel, D. "Biting the Silver Bullet — Toward a Brighter Future for System Development." *www.scs.carleton.ca.* 1992. http://www.scs.carleton.ca/~francis/Courses/314/Papers/BitingTheSilverBullet.pdf (accessed Enero 2009, 3).
16. Holzner, Steve. *Dojo Toolkit: Visual QuickStart Guide.* Berkeley: Peachpit Press, 2008.
17. Jauhar, Ali, and Tanaka Jiro. "Converting Statecharts into Java Code." *www.iplab.cs.tsukuba.ac.jp.* 2001. http://www.iplab.cs.tsukuba.ac.jp/paper/international/ali-idpt99.pdf (accessed Abril 2009, 18).
18. Kim, Dae Dae-Kyoo. "Role Based Metamodeling Language (RBML)." *www.secs.oakland.edu.* 2004.

http://www.secs.oakland.edu/~kim2/papers/RBML.pdf (accessed Marzo 2009, 23).
19. L.M.F. Carneiro, D.D. Cowan, C.J.P. Lucena. *ADVcharts: a Visual Formalism for Highly Interactive.* New York: ACM , 1994.
20. Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Upper Saddle River, New Jersey : Prentice Hall , 2004.
21. Linaje, M., Juan C. Preciado, and F. Sánchez-Figueroa. *A Method for Model Based Design of Rich Internet Application Interactive User Interfaces.* Heidelberg: Springer Berlin, 2007.
22. Mandel, Luis. *FORSOFT Project A2.* 1999. http://projekte.fast.de/Projekte/forsoft/welcomeE.html (accessed Diciembre 2008, 8).
23. Microsoft Corporation. "Rich Internet Application Architecture Guide." *www.microsoft.com.* 2008. http://www.microsoft.com (accessed marzo 13, 2009).
24. Niaz, A., and J. Tanaka. "Mapping UML Statecharts to Java Code." *www.actapress.com.* 2004. http://www.iplab.cs.tsukuba.ac.jp/paper/international/niaz_se2004.pdf (accessed Enero 2009, 28).
25. Riecke, Craig, Rawld Gill, and Alex Russell. *Mastering Dojo: JavaScript and Ajax Tools for Great Web Experiences (Pragmatic Programmers).* Lewisville: Pragmatic Bookshelf , 2008.
26. Rossel, Pedro, María Cecilia Bastarrica, and Ricardo Contreras. "GRAPHIC SPECIFICATION OF ABSTRACT DATA TYPES." *REVISTA FACULTAD DE INGENIERÍA, U.T.A. (CHILE)*, 2004: VOL 12 N°1, 2004, pp. 15-23.
27. Schwabe, Daniel, and Gustavo Rossi. "An Object Oriented Approach to Web-Based Application." *www.kcweb.org.uk.* 1998. http://kcweb.org.uk/weblibrary/OOWebAplDesign.pdf (accessed Enero 2009, 19).
28. —. "Developing Hypermedia Applications using OOHDM." *www.inf.puc-rio.br.* 1998. http://www.inf.puc-rio.br/~schwabe/papers/ExOOHDM.pdf.gz (accessed Noviembre 2008, 2).
29. Snow, Mano Marks and Kelly. "Methodology for Developing Web Design Patterns." *www.whitepapers.zdnet.com.* 2006. http://whitepapers.zdnet.com/abstract.aspx?docid=282586 (accessed Julio 2008, 1).
30. Staley, Tad. "Planning for RIA success." *www.adobe.com.* 2007. http://www.adobe.com/devnet/flex/articles/planning_ria/planning_ria.pdf (accessed Agosto 2008, 8).
31. Susana Montero, Paloma Díaz, and Ignacio Aedo. "Formalization of web design patterns using ontologies." *www.peterpan.uc3m.es.* 2003. http://peterpan.uc3m.es/pdfs/MonteroAwic03.pdf (accessed Septiembre 2008, 9).
32. Toxboe, Anders. *UI Patterns - User Interface Design Pattern Library.* 2004. http://ui-patterns.com/ (accessed Octubre 2008, 18).
33. Tuli, Amit. "IBM." 2005. http://www.ibm.com/developerworks/java/library/j-jwebunit/ (accessed Julio 2008, 14).
34. Urbieta, Matias, Gustavo Rossi, Jeronimo Ginzburg, and Daniel Schwabe. "Designing the Interface of Rich Internet Applications." *www2.computer.org.*

2007. http://www2.computer.org/portal/web/csdl/doi/10.1109/LA-Web.2007.14 (accessed Agosto 2008, 7).
35. Wallace, Nathan. "Design Patterns in Web Programming." *www.e-gineer.com.* 2000. http://www.e-gineer.com/v1/articles/design-patterns-in-web-programming.htm (accessed Octubre 2008, 28).
36. www.jsunit.net. *Sitio oficial de JsUnit.* http://www.jsunit.net (accessed Abril 2009, 9).
37. www.welie.com. *Welie.com Patterns in interaction design.* 2008. http://www.welie.com/ (accessed Enero 2009, 6).
38. www.wikipedia.org. *Enciclopedia comunitaria.* 2007. http://es.wikipedia.org (accessed Abril 2008, 7).
39. www.yahoo.com. *Design Pattern Library.* 2007. http://developer.yahoo.com/ypatterns/ (accessed Febrero 2008, 8).
40. Zakas, Nicholas C., Jeremy McPeak, and Joe Fawcett. *Professional Ajax, 2nd Edition (Programmer to Programmer).* Corporate Headquarters - Hoboken: Wrox - John Wiley & Sons, 2007.