Universidad de las Ciencias Informáticas

Facultad 4

Título: Estándares de soluciones a los problemas de presentación en el Polo de Sistemas Tributarios y Aduanas

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor: Mario Alberto Alvarez García

Tutor: Msc. Julio Cesar Diaz Vera

Junio del 2009

DECLARACIÓN DE AUTORÍA

Yo, Mario Alberto Alvarez García, me declaro como autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Mario Alberto Alvarez García

_____

Firma del Autor

Msc. Julio Cesar Diaz Vera

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor: Msc. Julio Cesar Díaz Vera, Graduado de Ing. en Telecomunicaciones y Electrónica en el año 2003 en la universidad central de Las Villas, se titula como Máster en Gestión de Proyectos en la UCI en el año 2007, Participo en el 1er Taller de Minería de datos Louisiana – Cuba en el año 2005, ha participado como ponente en las dos ediciones de UCIENCIA publicando en las memorias de ambos eventos.

AGRADECIMIENTOS

Más que a nada en este mundo, agradezco a mi mamá por todo el amor, apoyo y comprensión que siempre me ha dedicado. Ella es la persona que siempre ha estado incondicionalmente a mi lado y mi principal inspiración.

También a mi papá por conversar conmigo y orientarme siempre por el camino correcto.

Al "calvito" por su gran paciencia y ayuda, y por haberme tendido la mano en tantas ocasiones, algo que no podré olvidar nunca. Muchas gracias Julio.

A Yasmary, por haber dedicado tanto de su vida a mí y por tener tanta fe en mí, y por los inolvidables momentos que pasamos juntos.

También quiero agradecer a todos aquellos que de una forma u otra ayudaron a que yo llegará a ser lo que soy ahora.

DEDICATORIA

Este trabajo está dedicado a mi mamá por ser la persona que me valora más en este mundo. También a mi papa, y mis tres hermanitos Daguito, Alexander y Katherine.

RESUMEN

Este trabajo presenta una propuesta de procedimiento de desarrollo para el polo productivo sistemas tributarios y de aduanas, específicamente para la implementación de la capa de presentación. Basada en el uso de componentes y centrada en la arquitectura de línea base definida para el mismo.

La idea subyacente es aumentar el desempeño de los trabajadores de la capa de presentación y disminuir la curva de aprendizaje del framework de presentación utilizado en los proyectos del polo productivo.

La solución planteada propone el uso de algoritmos de selección de componentes a implementar del tipo “primero el de más frecuencia de uso”. Un mecanismo de identificación de necesidades de componentes sobre la base de los requerimientos de interfaz descritos en los casos de uso. El uso de patrones de diseño, estándares de codificación y el uso de mejores prácticas identificadas a lo largo de tres años de desarrollo en este entorno. Por último se define un set reducido de metadatos que son capaces de describir los componentes y que pueden ser utilizados por los desarrolladores para ubicar el componente que necesitan en cada momento.

TABLA DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1 INTRODUCCION

El Sistema Único de Aduanas (SUA) es la herramienta informática que emplea la Aduana General de la República (AGR) de Cuba para informatizar los diversos procesos aduanales. El SUA es elaborado por el Centro de Automatización para la Información y la Dirección en conjunto con la Universidad de la Ciencias Informáticas (UCI).

Con vistas a potenciar la usabilidad y el rendimiento del mismo, a partir del uso de prácticas de programación, probadas como aceptadas y formalizadas en patrones de diseño tanto arquitectónicos como a nivel de aplicación, y garantizar la separación física de la lógica del negocio de la presentación y la base de datos, su desarrollo asume el uso del paradigma de la Programación Orientada a Objetos (POO), la utilización del patrón de desarrollo web Modelo-Vista-Controlador (MVC) y de una capa de abstracción de datos para separar la lógica de la aplicación. (Cobo, 2008).

Específicamente el marco arquitectónico de desarrollo implica la utilización del framework de javascript Ext JS. Ext JS es una librería javascript para construir Aplicaciones de Internet Enriquecidas (RIA)[1]. Incluye componentes de interfaz gráfica configurables, de alto rendimiento, un modelo de componentes extensible y una API[2] intuitiva. Cuenta con dos licencias: una es Comercial y la otra Open Source (2009). La librería soporta los principales navegadores de internet empleados en la actualidad:

- Internet Explorer 6+
- Firefox 1.5+
- Safari 3+
- Opera 9+
- Chrome

[1] RIA es el acrónimo de Rich Internet Applications. Son un nuevo tipo de aplicaciones con más ventajas que las tradicionales aplicaciones Web. Esta surge como una combinación de las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales de escritorio.

[2] Application Programming Interface (Interfaz de programación de aplicaciones). Es el conjunto de funciones y procedimientos que ofrece una librería para ser utilizado por otro software como una capa de abstracción.

El polo de sistemas tributarios y de aduana (PSTA) viene trabajando con la definición de marco arquitectónico propuesta en (Cobo, 2008) desde aproximadamente 18 meses, en este tiempo y específicamente en la capa de presentación se han evidenciado los siguientes resultados.

Capacitación

Se han impartido tres ediciones del curso de capacitación en el dominio de la herramienta a un total de 48 desarrolladores de los distintitos proyectos del polo ST. Del total que recibió capacitación, solo el 48% trabaja actualmente como implementador de interfaz de usuario.

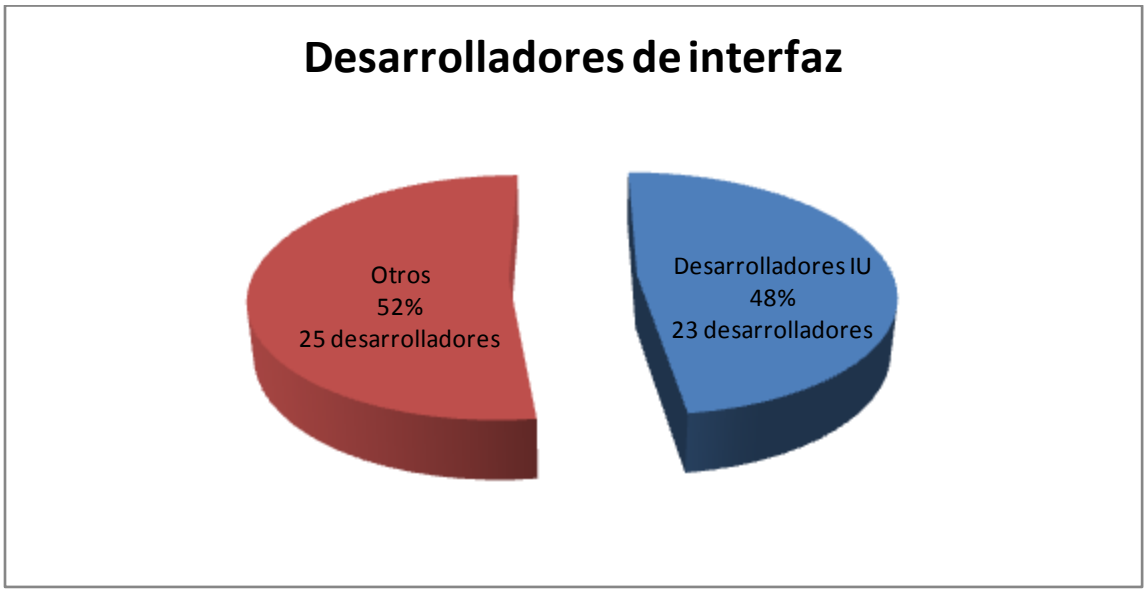

Figura 1 Desarrolladores que trabajan en la capa de presentación.

En la figura 2 está representada la distribución de desarrolladores por proyecto así como aquellos que han recibido tareas de implementación y una valoración tomada a partir del desempeño de los mismos acerca de su nivel de competencias para desempeñar el rol. En la referida figura se evidencia claramente que:

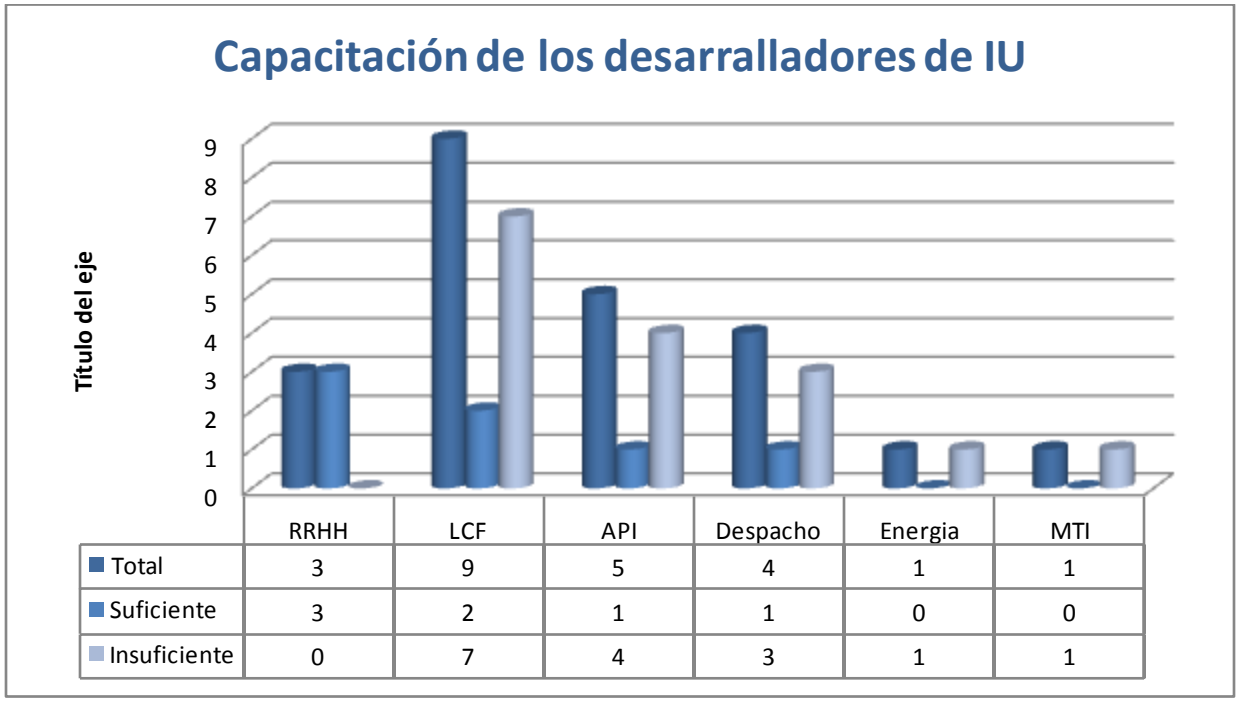

Figura 2 Estado de la capacitación de los desarrolladores de interfaz de usuario.

El Sistema de Gestión de Recursos Humanos (RRHH) es el único que cuenta con el total de sus desarrolladores capacitados para realizar su trabajo. Lucha Contra el Fraude (LCF) presenta una situación crítica, sólo el 18% de los desarrolladores tienen dominio de la librería, el 82% restante no cuenta con suficientes conocimientos para desempeñar su trabajo.

De cinco desarrolladores de interfaz que hay asignado al proyecto de Información Avanzada de Pasajeros (API), han iniciado acciones sólo el 20%. Energía y Medios de Transporte Internacional (MTI), cada uno tiene asignado un desarrollador que no domina el framework y en Despacho Comercial el 75% de los programadores de interfaz de usuario demuestran competencias mínimas para el desarrollado.

Solo el 30,4% de los desarrolladores de interfaz de usuario del polo han recibido tareas de desarrollo, el 59,6% restante aun no demuestra capacidades mínimas para desarrollar.

Reutilización de código

La tabla 1.1 muestra un resumen de los requerimientos de interfaz necesario para implementar un grupo de los casos de uso del sistema. Haciendo notar la reiteración de los mismos a lo largo del sistema. Es necesario señalar que hay requerimientos que son comunes a 5 casos de uso (Capturar y mostrar datos de contacto entre otros). Cada uno de los casos de uso ha sido implementado de manera independiente por desarrolladores diferentes y con soluciones particulares.

Módulo	Casos de Uso	Requerimientos de Interfaz
Calendario	CUCrearExcepcion	• Capturar fecha inicio y fecha fin • Capturar patrón y rango de recurrencia
	CUCrearRegla	• Capturar fecha inicio y fecha fin • Capturar patrón y rango de recurrencia
Estructura	CUCrearUnidad	• Capturar y mostrar datos de la unidad (nombre, objetivos, código de la unidad, categoría) • Capturar y mostrar datos de contacto
	CUCrearSubUnidad	• Capturar y mostrar datos de la subunidad (nombre, objetivos) • Capturar y mostrar datos de contacto
	CUCrearUnidad Organizativa	• Capturar y mostrar datos de la unidad organizativa (nombre, objetivos) • Capturar y mostrar datos de contacto
	CUCrearGrupoTrabajo	• Capturar y mostrar datos del grupo de trabajo (nombre, objetivos) • Capturar y mostrar datos de contacto
	CUCrearPlaza	• Capturar y mostrar datos de la plaza (código, usuario) • Capturar y mostrar datos de contacto

Trabajadores	CUCrearDocumento Identificacion	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CUCrearAnexo	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CURenovarContrato	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CUCrearFormalizacion delaDesignacion	• Capturar datos de los pagos adicionales • Capturar datos de tiempo mínimo mensual
	CUCrearContrato Adiestramiento	• Capturar datos de los pagos adicionales • Capturar datos de tiempo mínimo mensual
	CUCrearContratoPor PeriodoaPrueba	• Capturar datos de los pagos adicionales • Capturar datos de tiempo mínimo mensual
	CUCrearContratoPor TiempoDeterminado	• Capturar datos de los pagos adicionales • Capturar datos de tiempo mínimo mensual
Seleccion	CUCulminarFicha	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CUResultadosAnalisis Psicologico	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CUResultadosPrimera Entrevista	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
	CUResultados Verificacion	• Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)
Evaluacion del Desempeño	CUCrearEvaluacion Anual	• Tabla con datos de los trabajadores que serán evaluados • Información de la persona a evaluar • Tabla para asignar la evaluación a

		dicha persona
	CUCrearEvaluacion Mensual	• Tabla con datos de los trabajadores que serán evaluados • Información de la persona a evaluar • Tabla para asignar la evaluación a dicha persona
	CUCrearEvaluacion Trimestral	• Tabla con datos de los trabajadores que serán evaluados • Información de la persona a evaluar • Tabla para asignar la evaluación a dicha persona
	CUOpinar	• Información de la persona a evaluar
	CURevisarEvaluacion Mensual	• Información de la persona a evaluar
	CUVerEvaluaciones	• Información de la persona a evaluar

Tabla 1 Requerimientos de interfaz de los casos de uso por módulo.

Cumplimiento de los cronogramas de desarrollo

La tabla 2 muestra el cronograma de desarrollo de las interfaces de un grupo de casos de uso de sistema que pertenecen a RRHH. Sólo han sido computados los resultados de este subsistema debido a ser el único que cuenta con cronogramas detallados de desarrollo de interfaces en el período en que se desarrolla la presente investigación.

Dicha evidencia un desfasaje importante en cuanto a las fechas de culminación de las tareas con un término de 3.72 días de incumplimiento como promedio.

Caso de uso	Fecha planificada	Fecha concluyó
CUModificarDatos	20/enero/2008	24/enero/2008
CUPendientesaAceptacion	22/enero/2008	25/enero/2008
CUResultadosVerificacion	26/enero/2008	28/enero/2008
CUCrearFicha	29/enero/2008	5/febrero/2008
CUCrearUnidad	15/febrero/2008	19/febrero/2008
CUConfeccionarModificarEstructura	17/febrero/2008	20/febrero/2008
CUAsignarPatrimonio	19/febrero/2008	21/febrero/2008
CUInsertarTrabajador	23/marzo/2008	25/marzo/2008
CURenovarContrato	24/marzo/2008	30/marzo/2008
CUCrearContratodeAdiestramiento	3/abril/2008	10/abril/2008
CUReportarPendientesAnexo	6/abril/2008	7/abril/2008

Tabla 2 Cumplimiento de los cronogramas de desarrollo.

Resumen de los resultados

Los datos anteriormente expuestos permiten inferir los siguientes puntos:

- Los mecanismos de capacitación implementados reflejan una curva de aprendizaje elevada que no garantiza las competencias necesarias para desarrollar este rol.
- Los mecanismos de desarrollo no promueven la reutilización de código, implementándose una y otra vez los mismos requerimientos de interfaz.
- De manera empírica se ha constatado que los desarrollos realizados son prácticamente imposibles de mantener por otro desarrollador que no estuvo directamente involucrado en el mismo.
- No hay correspondencia entre los cronogramas de desarrollo y la terminación de las tareas.

Formulación del problema

En los acápites anteriores se han expuesto un grupo de elementos negativos que se han evidenciado durante el desarrollo de los distintos proyectos asociados al PSTA. Esta situación crea un marco favorable para desarrollar trabajos investigativos orientados a minimizar la repercusión de los mismos en los compromisos pactados. De manera particular el presente pretende abordar esta temática centrándose en el siguiente problema *¿Cómo aumentar el desempeño de los desarrolladores de interfaz en el PSTA de la Universidad de las Ciencias Informáticas?*

Este trabajo pretende accionar directamente sobre el proceso de desarrollo de interfaces en los proyectos del polo y la idea principal que se considerará expresa lo siguiente: *si se dirige el desarrollo de interfaces en el PSTA hacia la generación de componentes, debidamente documentados e implementados sobre la base de un estándar de codificación, que resuelvan los problemas típicos de interfaz entonces se reducirá de manera notable la curva de aprendizaje, se facilitara el mantenimiento de las soluciones y disminuirá la repetición de código contribuyendo directamente al cumplimiento de los cronogramas de este tipo de tareas.*

Por lo antes expuesto, **los objetivos generales** de este trabajo son identificar y tipificar las soluciones a las necesidades técnicas de la capa de presentación en el PSTA.

El objeto de estudio se centra en la arquitectura de línea base del PSTA, específicamente en el framework EXT JS.

El **campo de acción** está enmarcado en la arquitectura de la capa de presentación del PSTA.

Como **resultado** del trabajo se pretende lograr:

- Identificación de los problemas típicos de de la capa de presentación en el PSTA.
- Codificación de las soluciones estándares para los problemas típicos.
- Confección de un manual para facilitar la capacitación de los desarrolladores en la codificación de estos problemas.

Para dar cumplimiento a estos resultados se determinó que era necesario cumplir las siguientes **Tareas**:

- Identificación de los problemas comunes.
- Codificación de los patrones de solución.
- Validación de los patrones de solución.
- Documentar los patrones de solución.

El presente documento consta de 3 capítulos:

Capítulo 1:

Es la introducción al trabajo y es donde se aborda la problemática actual que existe en el polo de Sistemas Tributarios y de Aduanas con respecto al desempeño en la producción de los desarrolladores de interfaces de usuario.

Capítulo 2:

Este capítulo está dedicado a un estudio sobre las mejores prácticas empleadas en la actualidad para el desarrollo de interfaces de usuario empleado framework de javascript

Capítulo 3:

Se propone una vía de solución a los problemas que motivan este trabajo.

Capítulo 2 MODELO CONCEPTUAL Y ESTADO DEL ARTE

El desarrollo de aplicaciones informáticas ha sufrido transformaciones drásticas propiciadas por los importantes avances tecnológicos acaecidos en las dos áreas fundamentales asociadas al mismo: las telecomunicaciones y las tecnologías asociadas a la web.

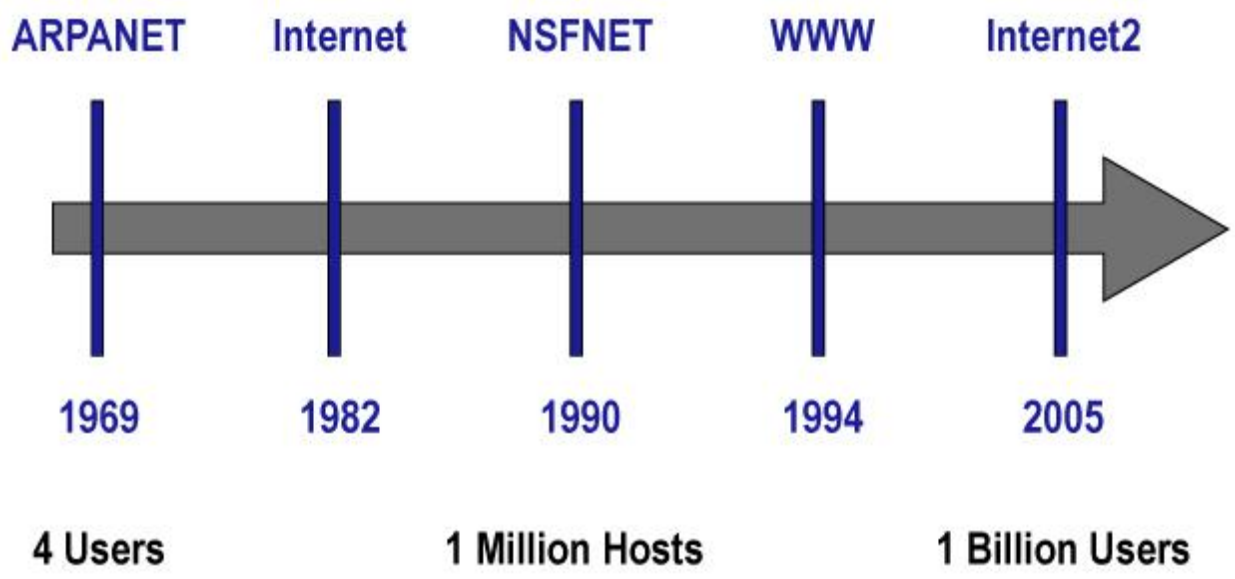

Figura 3 Línea de tiempo asociada a la evolución de Internet.

En el área de las telecomunicaciones, se destaca sobre manera la aparición de la ARPANET en el año 1969. ARPANET constituyó una solución a la problemática de interconectar los ordenadores, y así mismo marco los inicios de Internet, término que se empleó por primera vez al año siguiente.

Con ordenadores conectados entre sí, se hizo imperante estandarizar el intercambio de datos a través de la red, para garantizar que, sin importar el fabricante o el sistema operativo, la transferencia de los mismos pudiese ser efectuada.

Es así como surge a principio de la década del ‘70 el protocolo NCP (protocolo de control de

red), el cual por cuestiones operativas (no permitía incorporar redes que estuvieran fuera de ARPANET, por lo que carecía de control de errores), fue reemplazado en la siguiente década por el protocolo TCP/IP (protocolo de control de transmisiones/protocolo de internet). Con la entrada de TCP/IP la red alcanzó un entorno de arquitectura abierta. (Lynch, y otros, 2003).

El protocolo TCP/IP es la base de Internet, y sirve para enlazar computadoras que utilizan diferentes sistemas operativos, incluyendo PC, minicomputadoras y computadoras centrales sobre redes de área local (LAN) y área extensa (WAN). TCP/IP fue desarrollado y demostrado por primera vez en 1972 por el Departamento de Defensa de los Estados Unidos.

Otro punto determinante en esta área ha estado asociado al aumento de capacidad de las trasmisiones vinculado a la aparición de la fibra óptica y el despegue de las tecnologías xDSL.

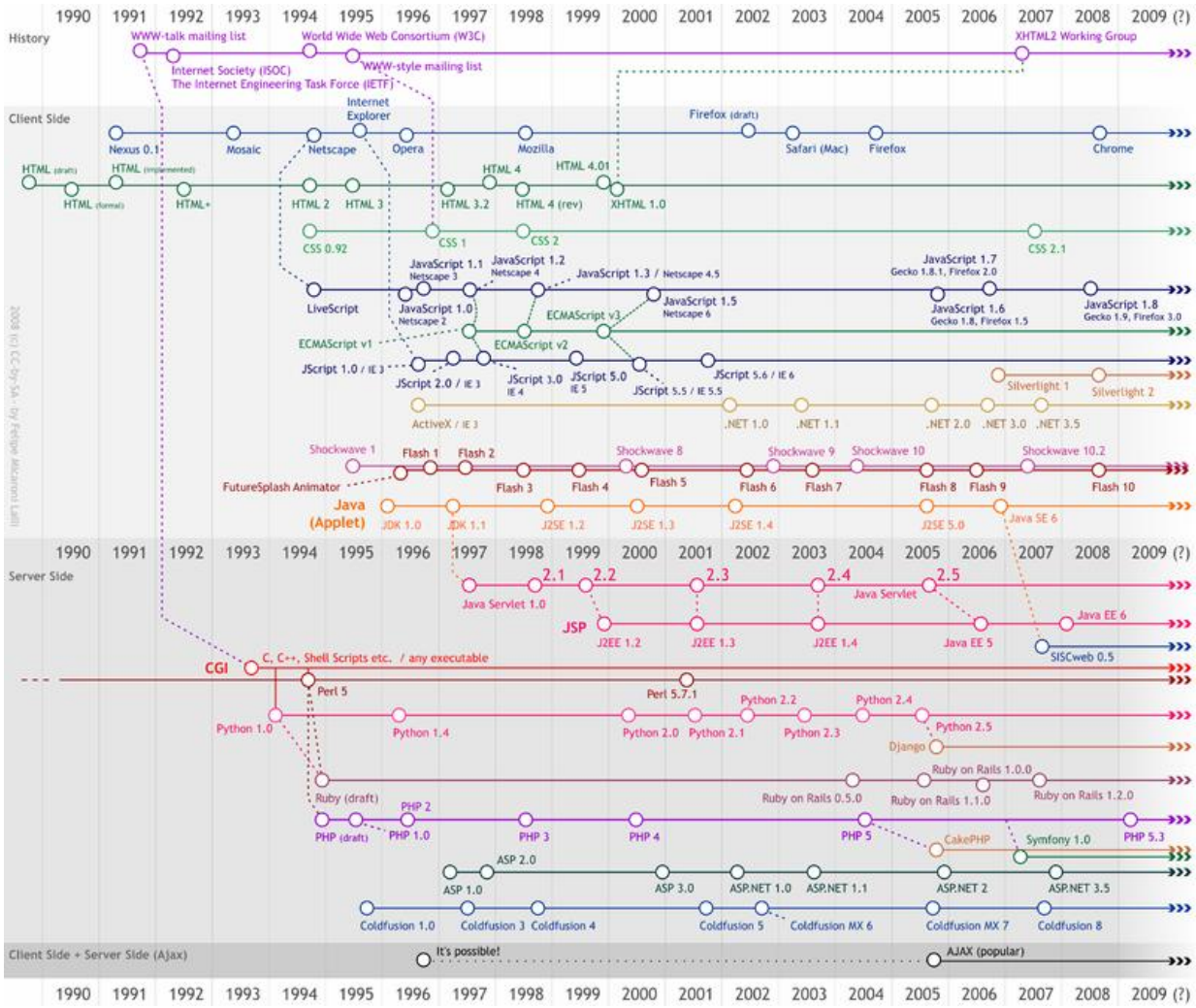

Figura 4 Línea de tiempo asociada a la evolución de las tecnologías web.

En el área de las tecnologías que están asociadas a la web, la palabra Web hace referencia al conjunto abstracto de información que podemos encontrar en el espacio virtual formado por Internet. La base de Internet es un conjunto de protocolos, normas y lenguajes por los cuales se comunican los ordenadores. World Wide Web es un conjunto de protocolos que permiten compartir información a través de Internet (Bernaus, y otros, 2000).

Se destacan por su importancia HTML, un lenguaje de composición de documentos y especificación de ligas de hipertexto que define la sintaxis y coloca instrucciones especiales

que no muestra el navegador web[3], aunque si le indica cómo desplegar el contenido del documento, incluyendo texto, imágenes y otros medios soportados. El lenguaje HTML (HyperText Markup Languaje) es utilizado para crear las páginas de información que podemos ver cuando navegamos por la Web. (Musciano, et al., 1999).

El lenguaje HTML es un subconjunto del lenguaje SGML (Standard Generalized Markup Language – Lenguaje de Marcado Generalizado), el cual consiste en un estándar internacional para la definición de métodos para representar texto en aplicaciones electrónicas (Sperberg-McQueen, et al., 1994).

Conforme la navegación web ganaba popularidad, las páginas realizadas con HTML crecieron en complejidad y tamaño y la mayoría de los internautas poseían conexiones con muy escaso ancho de banda. En estas circunstancias, resultaba fastidioso tener que hacer múltiples peticiones al servidor para realizar simples validaciones. Es así como nace la necesidad de contar con un lenguaje que se ejecutase en el lado del cliente y que disminuyera el número de peticiones realizadas al servidor. Nestcape fue el primero en desarrollar un lenguaje con estas características, inicialmente llamado LiveScript, y que posteriormente paso a ser Javascript.

Javascript es un lenguaje ligero, el no funciona por sí solo, pero que fue diseñado para ser embebido fácilmente en otros productos y aplicaciones, como son los navegadores web (Powell, y otros, 2004).

El lenguaje Javascript se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como aparición y desaparición de texto, animaciones, acciones que se activan al pulsar botones u otros elementos y ventanas con mensajes de aviso al usuario. Técnicamente, Javascript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras

[3] Un navegador web es un programa que permite visualizar la información que contiene una página web. El navegador interpreta el código, HTML generalmente, en el que está escrita la página web y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos.

palabras, los programas escritos con Javascript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios (Eguíluz, 2008).

Este lenguaje de programación tiene algunas limitaciones:

- Javascript no es un lenguaje orientado a objetos.
- No se pre compila.
- No es obligatorio declarar las variables.
- Verifica las referencias en tiempo de ejecución.
- No tiene protección del código, ya que se descarga en texto claro.
- Es un lenguaje interpretado, o sea, que el sistema lo lee y traduce al mismo tiempo que lo va ejecutando

Otro avance tecnológico fue el lenguaje CSS (Cascading Style Sheets – Hojas de Estilo en Cascada), que consiste en un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir.

Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. CSS se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. Los *Estilos* definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca ese elemento (W3C, 2008).

AJAX (Asynchronous JavaScript + XML – Javascript Asíncrono + XML) y el objeto XMLHttpRequest marcaron un paso de avance vital en las tecnologías de desarrollo web. El término AJAX se acuñó por primera vez en el artículo “Ajax: A New Approach to Web Applications” publicado por Jesse James Garrett el 18 de Febrero de 2005 y se define de la siguiente forma: “Ajax no es una tecnología en sí mismo. En realidad, se trata de la unión de varias tecnologías que se desarrollan de forma autónoma y que se unen de formas nuevas y sorprendentes.”

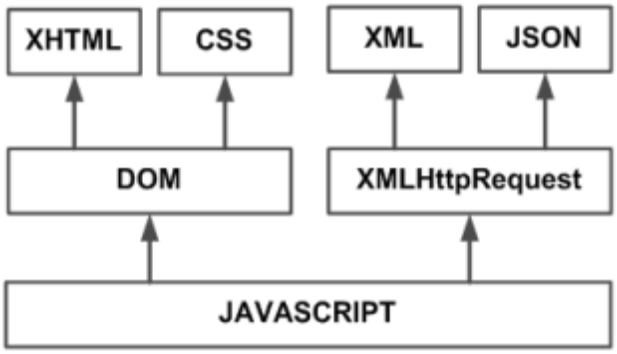

Figura 5 Tecnologías agrupadas bajo el concepto Ajax (Eguiluz, 2007).

En el desarrollo clásico tanto de aplicaciones como sitios web la comunicación con el usuario es síncrona respecto de las interacciones del usuario, es decir:

1. El usuario realiza una petición al servidor.
2. El servidor envía la página solicitada.
3. El usuario comienza a “leer” la página.
4. Llega un momento dado en el cual el usuario desea cambiar de página y mediante formularios o links realiza una petición al servidor volviendo al paso número 1.

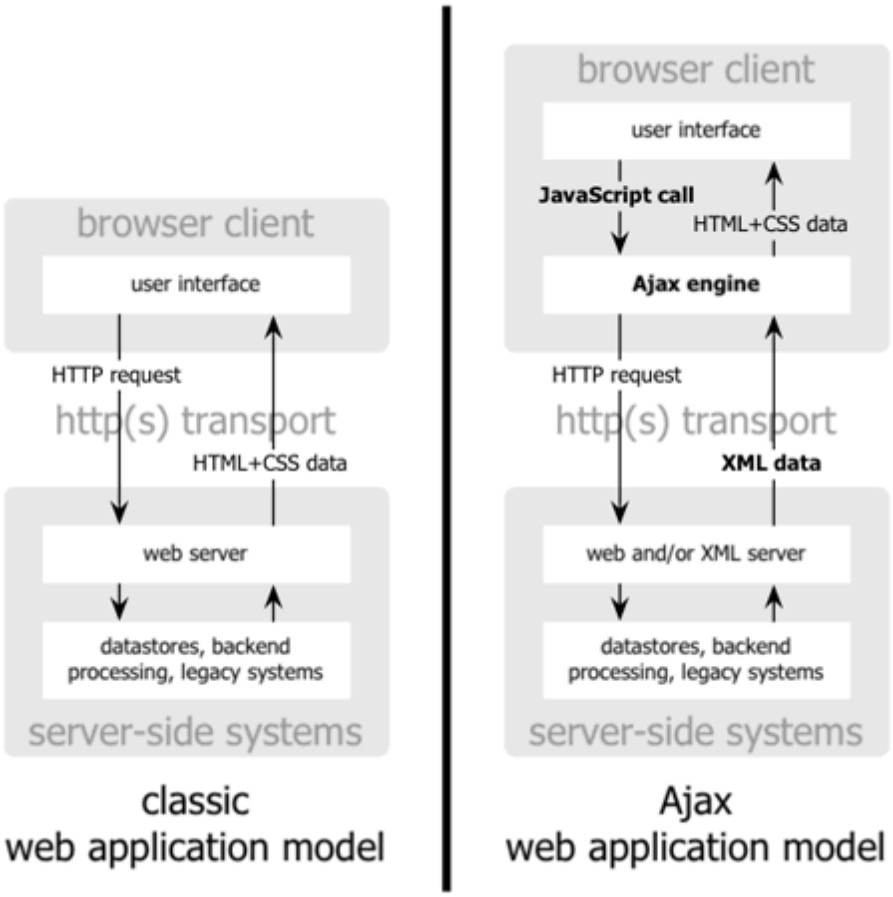

Figura 6 imagen de la izquierda describe el modelo tradicional y la imagen derecha describe el modelo Ajax (Eguíluz, 2007).

En cambio la comunicación asíncrona que provee AJAX no implica sincronismo ante los eventos del usuario, sino que ante cualquier evento del usuario nosotros podemos proceder en consecuencia, es decir:

1. El usuario realiza una petición al servidor.
2. El servidor envía la página solicitada.
3. El usuario comienza a "leer" la página.
4. Llega un momento dado en el cual el usuario quiere cambiar de página o alterar la información que contiene la misma. En ese momento dado la aplicación teniendo definidos los eventos posibles actúa en consecuencia a la interacción que haya podido suceder. Entonces podremos cambiar al usuario de página, o por el contrario modificar la misma para satisfacer al usuario. Estos cambios no implican cambiar de página.

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

La siguiente figura muestra la diferencia más importante entre una aplicación web tradicional y una aplicación web creada con AJAX.

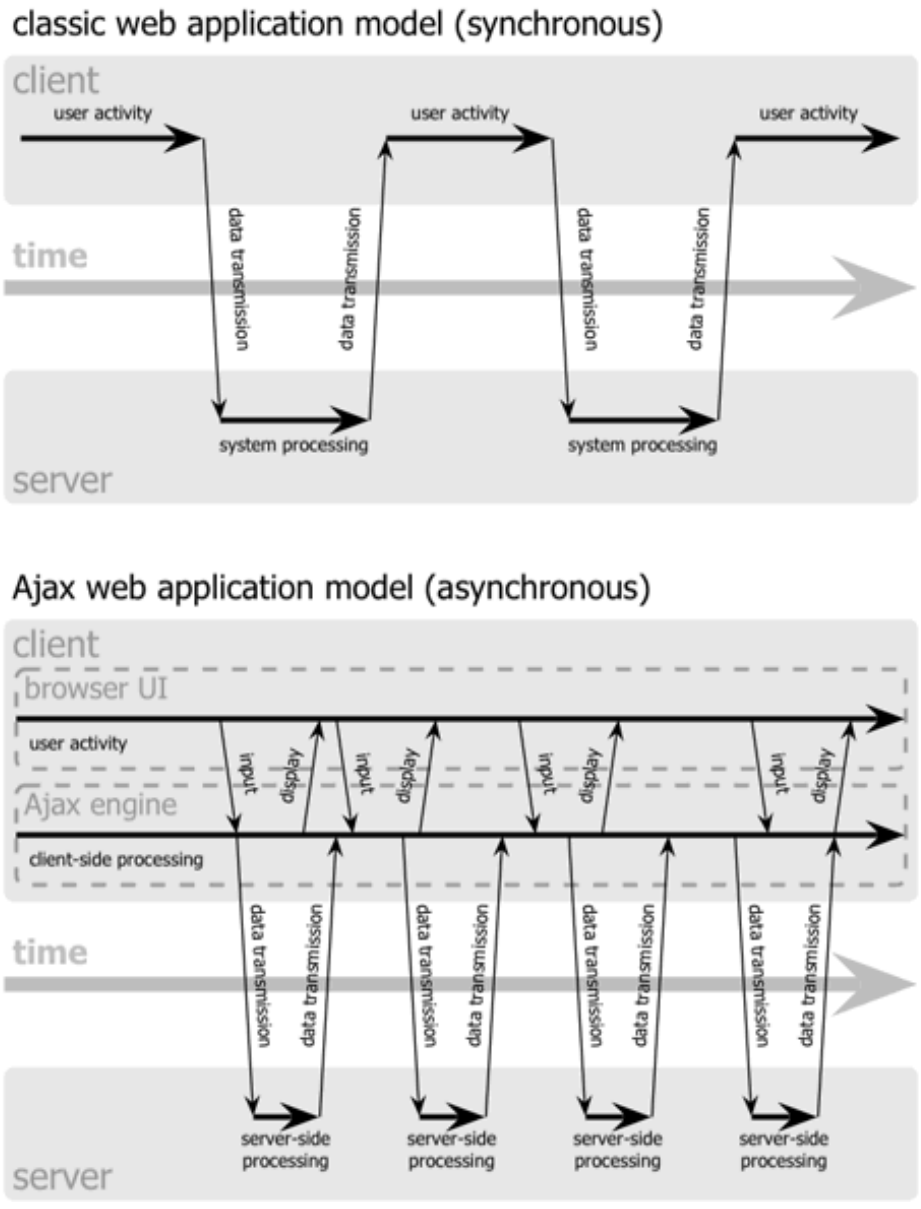

Figura 7 Imagen superior: Interacción síncrona. Imagen inferior: Interacción Asíncrona (Eguiluz, 2007).

Las peticiones HTTP al servidor se transforman en peticiones Javascript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción del servidor requiere la respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor (Eguiluz, 2007).

XML (Extensible Markup Languange – Lenguaje Extensible de Marcas) es una simplificación y adaptación del lenguaje SGML. El lenguaje XML es empleado como un estándar para el intercambio de información estructurada entre diferentes plataformas. Puede ser usado una gran variedad de aplicaciones, tales como bases de datos, editores de texto, hojas de cálculo, etc. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil (Montero, 2001).

Una alternativa al lenguaje XML para el intercambio de información que se realiza con AJAX, es JSON (Javascript Object Notation – Notación de Objetos de Javascript). JSON es un formato ligero para el intercambio de datos, fácil de leer para los humanos, e igualmente fácil de generar y parsear por los ordenadores. En Javascript, JSON puede ser analizado trivialmente usando un procedimiento propio del lenguaje, lo cual ha sido fundamental para la aceptación de JSON por parte de la comunidad de desarrolladores Ajax (Zammetti, 2007).

La notación JSON está constituida por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

La evolución de estas tecnologías ha provocado una transición de aplicaciones tradicionales hacia aplicaciones que funcionan a través de la web, enfocada a los usuarios finales. Se trata de aplicaciones que generen colaboración y servicios que reemplacen las aplicaciones de

escritorio. La representación de esta evolución es conocida como Web 2.0, término que se dio a conocer en el año 2004 cuando se realizó la primera Conferencia sobre la Web 2.0.
Web 2.0 más que un desarrollo tecnológico innovador, es una reorganización y una nueva concepción de lo que hay en Internet y de lo que se está construyendo. Los sitios web diseñados bajo la concepción 2.0 están hechos bajo los preceptos de la usabilidad y bajo la idea de tener a la mano un mayor número de recursos (O'Reilly, 2005).

Una Aplicación Enriquecida de Internet (RIA – Rich Internet Application) es una aplicación que se ejecuta en un navegador web, la cual emplea una capa intermedia que elimina tener que refrescar la página, procedimiento común en la mayoría de aplicaciones web. Las herramientas más comunes con las que se pueden lograr esta capa intermedia son Javascript y AJAX. En una RIA se implementan técnicas que permitan al usuario experimentar que esta trabajado en una aplicación tradicional de escritorio. Algunas de estas técnicas consisten en transferencia asíncrona de datos, arrastrar-soltar elementos visuales (Tretola, 2008).

El auge de las RIA, trajo consigo el desarrollo frameworks de javascript que implementaran estas técnicas dentro de ello se destacan Dojo Toolkit, Ext JS, Prototype, JQuery[4], etc.

Un framework es una estructura de software compuesta de componentes, personalizables e intercambiables, para el desarrollo de una aplicación y puede ser considerado como una aplicación genérica incompleta y configurable, a la que se le pueden añadir las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones (Gutiérrez, 2005).

Un framework javascript puede definirse como un conjunto de componentes que componen un diseño reutilizable que facilita y agiliza el desarrollo de aplicaciones basadas en javascript, especialmente para Ajax.

[4] Frameworks de javascript que facilitan el desarrollo de las aplicaciones web basadas en Ajax y que implementan componentes reusables.

Según plantea (Mili, y otros, 1995), la reutilización es la única vía para desarrollar sistemas de información que tengan los niveles de calidad exigidos, dentro de las restricciones existentes de tiempo y presupuesto.

El concepto de reutilización está concebido como la combinación de componentes de código almacenados en una biblioteca, y usados cuando estos sean necesarios (Guarachi, 2005). Entre los beneficios que aporta la misma, se destaca que mejora la productividad de la empresa, pues disminuye el tiempo de desarrollo y de costes. Por otra parte, garantiza un software de mayor fiabilidad y eficiencia, aunque al principio pueda creerse que no, lo que asegura un producto de mejor calidad (Lopez, 2006).

Los trabajos relacionados con la reutilización según el objeto y el método de reutilización usado pueden ser clasificados como:

- Reutilización de código.
- Reutilización de diseños.
- Reutilización de especificaciones.

La reutilización de código es la más común y extendida, la cual se presenta en la fase de implementación. Ejemplos de lo que se puede reutilizar en esta fase son: código fuente, código objeto, bibliotecas estándar, etc.

La mantenibilidad es la facilidad con la cual puede ser modificado el código de un sistema frente a cambios en el ambiente, requerimientos funcionales o especificaciones funcionales. (Bengtsson, et al., 2000).

Escribir código con la idea en mente de que sea mantenible, tiende a dar como resultados un software robusto, tanto en fiabilidad como seguridad, a la vez que el código es más correcto. El simple hecho de prestar mayor atención a como el código luce, cuan claro es, cuan apropiados son los nombres de las constantes y variables, generalmente suele dar lugar a un mejor programa en general.

A excepción de algunos casos, el costo de vida empleado para escribir código sostenible desde el principio siempre será menor que el costo que es necesario para escribir código con poca o

ninguna reflexión sobre la mantenibilidad. Esto se debe a que un código claro y bien estructurado es siempre más fácil de comprender. Si se adiciona el factor de fiabilidad y seguridad, los beneficios obtenidos del código mantenible aumentan, no solo porque el programa será más sólido, sino también porque cuando el error este encubierto, encontrarlo y corregirlo será mucho más fácil.

En (Szyperski, 2002) se plantea que un componente es una unidad de composición de aplicaciones software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio.

Por su parte, (Ariza, et al., 2004) ofrece un concepto más abarcador sobre lo que es un componente, y expresando que es:

- Una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces.
- Una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Un componente puede ser desplegado independientemente y es sujeto a la composición de terceros.

Más allá de su definición existen algunas características claves para que un elemento pueda ser catalogado como componente:

- **Identificable**: Debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.
- **Puede ser remplazado por otro componente**: Se puede remplazar por nuevas versiones u otro componente que lo remplace y mejore.
- **Con acceso solamente a través de su interfaz:** Debe asegurar que estas no cambiaran a lo largo de su implementación.
- **Sus servicios no varían**: Las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación sí.

- **Bien Documentado**: Un componente debe estar correctamente documentado para *facilitar su búsqueda* si se quiere actualizar, integrar con otros, adaptarlo, etc.
- **Es genérico:** Sus servicios debe servir para varias aplicaciones.
- **Reutilizado dinámicamente:** Puede ser cargado en tiempo de ejecución en una aplicación.

Estas definiciones no son mutuamente excluyentes, por el contrario, se complementan y construyen el significado no sólo de componente, también el significado del Desarrollo de Software Basado en Componentes (DSBC).

El DSBC es una disciplina que se apoya en componentes de software ya desarrollados, que son combinados adecuadamente para satisfacer los requisitos del sistema. Construir una aplicación se convierte en la búsqueda y ensamblaje de piezas prefabricadas, desarrolladas en su mayoría por terceros (Leavens, et al., 2000).

El DSBC busca, dentro de otros objetivos, reducir el tiempo de trabajo, el esfuerzo que requiere implementar una aplicación y los costos del proyecto, y, de esta forma, incrementar el nivel de productividad de los grupos desarrolladores y minimizar los riesgos globales. Es preciso resaltar el hecho de que el DSBC pertenece al paradigma de programación de sistemas abiertos, los cuales son extensibles y tienen una interacción con componentes heterogéneos que ingresan o abandonan el sistema de forma dinámica, es decir que los componentes pueden ser remplazados, por otros independientemente de su arquitectura y desarrollo (Ariza, et al., 2004).

La mayoría de los frameworks de desarrollo java script hacen uso del desarrollo de software basado en componentes con vistas a aprovechar las ventajas planteadas anteriormente. Ext JS y Dojo Toolkit son de los ejemplos más significativos en este sentido y para este trabajo tiene especial importancia los mecanismos que propone Ext JS para realizar este tipo de desarrollo.

Ext JS posee un modelo de componentes jerárquico que facilita la reutilización de cada uno de los componentes que forman esta librería y promueven la extensión de los mismos con vista a desarrollos específicos de alto grado de personalización. Estas características hacen que encaje perfectamente dentro del modelo a desarrollar en PPSTA que pretende el desarrollo de

una arquitectura de dominio específico potencializando el desarrollo de componentes que encapsulen lógica de funcionamiento asociada a los requerimientos del dominio.

En la base de la estructura arquitectónica del framework se encuentra la clase Ext.util.Observable, que es utilizada como interfaz para la publicación de los diferentes eventos que poseen todas las clases y componentes que implementa la librería.

La clase Ext.Component constituye la base de todos los componentes que están presentes en Ext JS. Provee un ciclo para la creación, renderización y destrucción de los componentes. Ext.BoxComponent es una especialización de la clase componente, que permite ajustar automáticamente las dimensiones y el posicionamiento de los componentes.

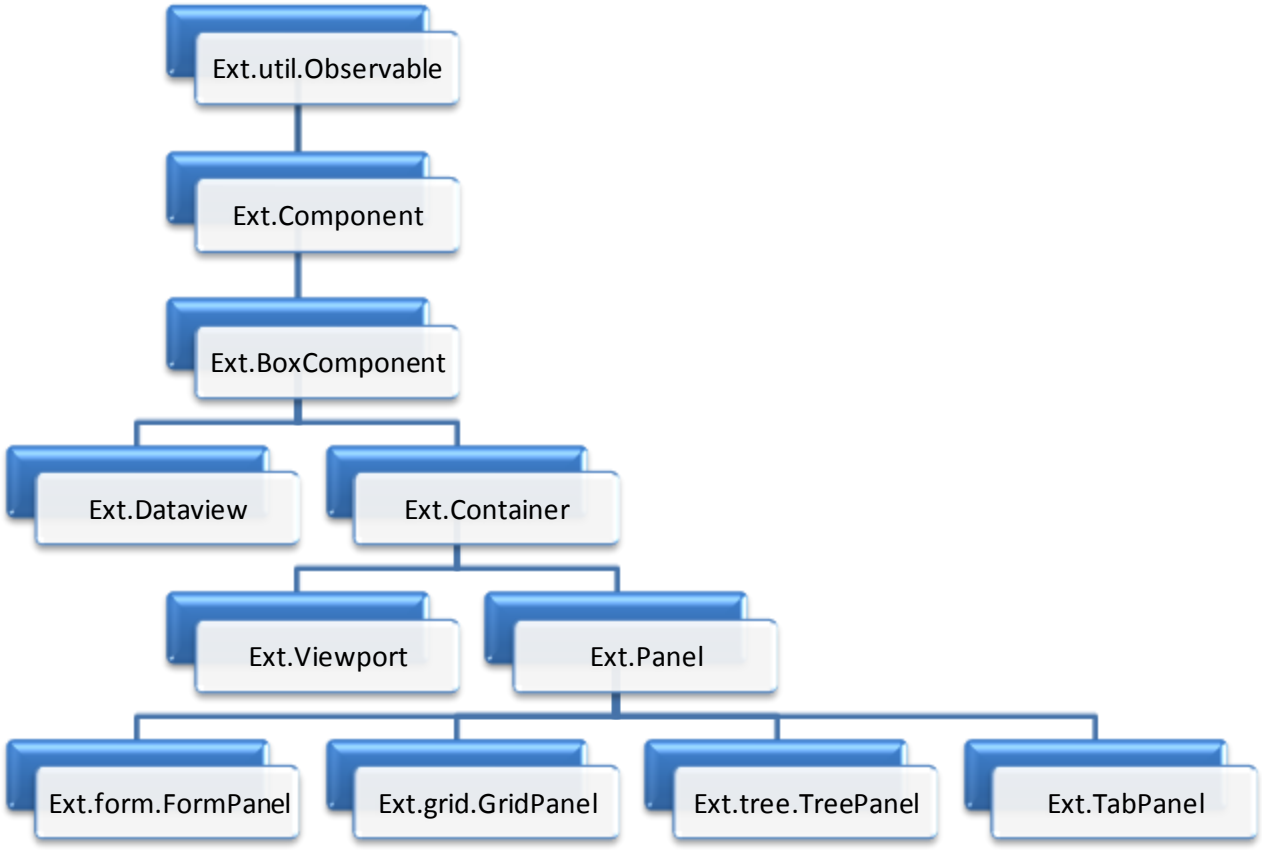

Figura 8 Estructura arquitectónica de Ext JS.

Los componentes de tipo Ext.Container, o que extienden del mismo, son empleados como contenedores de otros componentes. El contenedor que es usado con mayor frecuencia es Ext.Panel, el cual tiene funcionalidades específicas que hacen que sea perfecto para la construcción de interfaces. El componente Ext.Viewport es contenedor especial, que automáticamente sincroniza sus dimensiones con el navegador, es por esto que solo puede existir solo una instancia del mismo por cada página.

Ext.form.FormPanel es el componente que contiene el framework para la creación de formularios, y que sirve de contenedor para los elementos que contienen los formularios, como campos de texto, listas desplegables, campos de fecha, etc. Los datos del formulario son transmitidos al servidor usando Ajax.
Ext.tree.TreePanel, es un componente especial que es empleado para mostrar información jerárquica y el componente Ext.grid.GridPanel se usa para mostrar información en forma de tabla. Ambos componentes definen un vasto grupo de funcionalidades, que permiten una interacción muy completa con el usuario.

La revisión realizada del desarrollo histórico asociada a la capa de presentación en arquitecturas web muestra una alta tendencia a la utilización de frameworks de desarrollo java script y dentro de estos son especialmente utilizados aquellos que siguen arquitecturas de desarrollos basadas en componentes. Las razones por las que este tipo de desarrollos se han hecho tan populares están asociadas a hechos aceptados como “verdades absolutas” por parte de la comunidad empresarial y académica y que plantean lo siguiente:

- Disminuye el tiempo de desarrollo.
- Disminuye el esfuerzo necesario para realizar una tarea.
- Aumentar el rendimiento y la productividad de los desarrolladores
- Disminuir los costos de desarrollo.

Estos elementos pueden ser mapeados de manera natural con los problemas planteados como eje central de este trabajo y que están asociados al bajo desempeño de los desarrolladores de la capa de presentación en el polo productivo STA.

Con estos elementos en la mira entonces parece posible aplicar la tendencia a desarrollo por componentes al desarrollo de la capa de presentación del PPSTA esperando tal y como fue planteado en la hipótesis un aumento del rendimiento de los desarrolladores que repercuta en el cumplimiento de los cronogramas, el mantenimiento de las aplicaciones y la curva de aprendizaje.

Capítulo 3 SOLUCION PLANTEADA

Introducción

En este capítulo se expone el procedimiento empleado en la detección de los problemas típicos de interfaces que se presentan en los proyectos que integran el Polo de Sistemas Tributarios y Aduanas (PSTA).

Procedimiento utilizado en la detección de los problemas

La identificación de los problemas típicos de interfaces que presentan en los proyectos del PSTA, se realizó siguiendo un grupo de tareas, que abarcan desde la identificación del problema hasta la documentación del mismo. En la figura 1 están representadas dichas tareas.

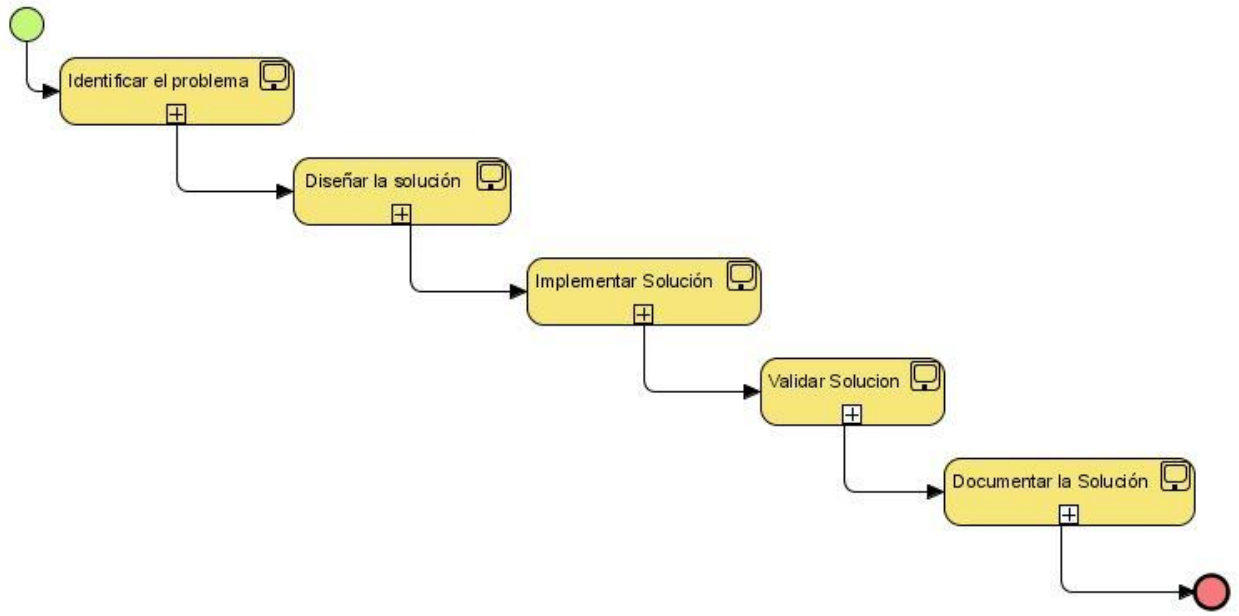

Figura 9 Procedimiento para la identificación de problemas en el PSTA.

Identificar el problema

La identificación del problema es llevada a cabo por los desarrolladores en el momento en que tienen que enfrentar un nuevo caso de uso, para lo cual deben estudiar cuidadosamente los requerimientos de la interfaz y buscar dentro del repositorio de componentes de interfaz alguno que le permita dar una solución total o parcial al problema. De no existir ningún elemento se desarrollaría y documentaria un nuevo componente que pasaría a formar parte del repositorio.

En el estado actual del PSTA esto es una utopía ya que no se ha materializado la figura del repositorio de componentes. Lo cual obliga a introducir una variación en este flujo de actividades orientado a desarrollar los componentes necesarios y siguiendo la política de implementar antes aquellos que tengan mayor probabilidad de ser reutilizados. El criterio seleccionado para ello se basa en PMF (primero el más frecuente), una adaptación de los algoritmos de remplazo de memoria y planificación del micro procesador en sistemas operativos, a través del cual se van a priorizar aquellos componentes que se hayan presentado mayor cantidad de veces en los casos de usos descritos hasta el momento.

Para ello se realizó una revisión a todos los casos de uso del sistema que han sido identificados, y se agruparon los requerimientos de interfaz que tenían en común y que podían ser implementados como componentes que pudiesen ser rehusados especificando además la frecuencia de aparición. El resultado de esta tarea se muestra en la tabla 3.

Requerimientos	Total de apariciones
Tabla con datos de los trabajadores (expediente, carnet identidad, nombre, segundo nombre, primer apellido, segundo apellido)	7
Información de la persona	6
Capturar y mostrar datos de contacto	5
Capturar datos de los pagos adicionales	4
Capturar datos de tiempo mínimo mensual	4
Capturar y mostrar datos del grupo de trabajo (nombre, objetivos)	3
Tabla para asignar la evaluación a una persona	3
Capturar fecha inicio y fecha fin	2

Tabla 3 Requerimientos y total de apariciones.

Diseño de la solución

Primeramente es necesario resaltar que debido a que javascript no es un lenguaje orientado a objetos, no maneja de manera explícita los conceptos de clase, componente y objetos. De cualquier forma e intentando acercarse a los estándares de la programación orientada a objetos, el lenguaje javascript incluye un mecanismo de "simulación de herencia" implementado a partir de la propiedad prototype.

Este mecanismo permite extender las funciones javascript dándole un comportamiento similar al de clases en la programación orientada a objetos. Por esta razón y en lo adelante en este trabajo será utilizado indistintamente el concepto de clase, componente u objeto para hacer referencia a los elementos javascript propios del framework que serán usados como base o resultado para implementar los requerimientos de visualización identificados.

Los nuevos componentes a desarrollar deberán ajustarse a la jerarquía planteada en la arquitectura del framework (ver figura 8). El mecanismo que se usará para ello consiste en extender los componentes nativos del framework, añadiendo las características deseables y creando componentes definidos por el usuario que podrán combinarse y extenderse para crear nuevos componentes a medida que el desarrollo de las aplicaciones lo requieran.

Cuando se extiendan los componentes se hará centrándose en añadir a los ya existentes un conjunto de campos definidos que poseen un comportamiento común y que se pueden emplear en diversos de casos de uso. O también en agregarle al componente nuevas funcionalidades.

Para este trabajo se define como mecanismo de extensión de componentes propuesto por los desarrolladores de Ext JS, el cual ha alcanzado un alto nivel de aceptación gracias a la simplicidad con que maneja esta tarea.

Debido a que javascript no sigue el paradigma de objetos, es imposible definir herencia de manera directa dentro de él. Por ello en Ext JS se define una función que provee un mecanismo para simular la herencia de clases, llamada Ext.extend().

Con esta función, se pueden modificar o extender las funcionalidades de los componentes existentes en el framework, sin hacer cambios directamente al código de los mismos. A continuación se presenta un ejemplo de cómo utilizar esta función:

```
var NuevoComponente = Ext.extend(Ext.Panel, {
    //Constructor del componente
    constructor: function (config) {
        // Aplicar a este objeto lo que esta en el objeto de configuracion
        Ext.apply(this, config);

        // Llamada al constructor de la clase padre
        NuevoComponente.superclass.constructor.call(this);
    },

    // Asi se añaden nuevas funciones al componente
    miNuevaFuncion: function() {

    },
    // Sobreescribir una funcion del componente
    onRender: function() {
        //Cuando se sobreescribe una funcion, es necesario llamar al metodo de la clase padre
        MyPanel.superclass.onRender.apply(this, arguments);
        //Un ejemplo de llamar la nueva funcion
        this.miNuevaFuncion();
    }
});
//Finalmente se registra el componente para poder usarlo a traves del xtype
Ext.reg('nuevocomponente', NuevoComponente);
```

Figura 10 Uso de la función Ext.extend()

- Cuando se hace uso de herencia en cualquier lenguaje orientado a objetos es definitorio la capacidad de invocar al constructor de la clase base. Javascript no provee un mecanismo para hacer esto, es por esto que hay que llamarlo explícitamente usando la propiedad "superclass[5]" y uno de los métodos "call" o "apply". El primer parámetro que recibirán los métodos call() y/o apply()[6] siempre será "this", para asegurar que el contexto de la ejecución este en el ámbito de la función invocadora.

[5] Propiedad que poseen las clases y los componentes de Ext JS, que puede ser usada para referenciar a la clase padre.

[6] Métodos nativos del lenguaje Javascript que permiten ejecutar una función sobre un objeto deferente que se le pasa como parámetro. Ambos métodos reciben dos parámetros,

- El primero consiste en una referencia al objeto sobre el cual ejecutar la función

- La función Ext.extend() recibe dos parámetros, el primero será siempre el componente del que se extenderán las funcionalidades.
- El segundo parámetro consiste en un objeto JSON que contiene las características del nuevo componente. Esto puede consistir en nuevas características o la redefinición de otras existentes en el componente.

El código de la figura 11 puede ser empleado como plantilla para la implementación de nuevos componentes. Lo primero que se hace es declarar Ext.extend(), la cual retorna una referencia a una función, que es almacenada en la variable "NuevoComponente", y que posteriormente es usada para crear instancias del nuevo componente. A continuación se muestra un ejemplo:

```
var primerNuevoComponente = new NuevoComponente({
    title:'Este es el primer componente que creo'
});
```

Figura 11 Plantilla para la creación de componentes.

En el objeto de configuración que recibe como segundo parámetro esta función, se redefinen las características de los componentes nativos y se declaran las nuevas que contendrá el nuevo componente.

El código precedente muestra el caso de la función constructor, que es el primer método que se ejecuta cuando se instancia el componente. Este método recibe un parámetro que contiene la configuración que tendrá el componente y que debe ser aplicado al objeto, mediante la función Ext.apply(). Después es llamado el constructor de la clase padre, para preservar las funcionalidades de la clase base.

En la función llamada miNuevaFuncion() puede ser encapsulado nuevas características del componente que está siendo implementado y pueden existir tantas como se requieran.

- El segundo difiere para ambos: call() recibe una lista de parámetros y apply() recibe un arreglo como parámetro. El método call() es favorecido sobre apply()**Fuente especificada no válida.**.

El método onRender() forma parte de una secuencia que siguen los componentes nativos de Ext JS cuando son instanciados. Este método en particular se ejecuta automáticamente en el momento que es dibujado el componente.

Como parte de la solución propuesta se utilizara el patrón template method (método de plantilla) que se encuentra implementado en Ext JS, el objetivo es simplificar la tarea asociada a redefinir los pasos que forman parte del ciclo de vida de los componentes. Para lograr esto sigue una secuencia específica para la generación de componentes que se muestra a continuación:

- **initComponent[7]:** El método más importante en el proceso de inicialización para las clases que extienden sus funcionalidades de los componentes nativos de la librería. Este método fue diseñado para contener la lógica del componente y así no tener que definir el constructor.
- **onRender:** Este método se ejecuta automáticamente antes de que el componente es dibujado en el navegador.
- **afterRender:** Se emplea para ejecutar alguna acción después que el componente ha sido dibujado.
- **beforeDestroy:** Cualquier acción a ejecutar antes de que el componente sea destruido.
- **onDestroy:** Es llamado cuando una instancia del componente es destruida. Es muy útil cuando el componente posee recursos de los que Ext JS no tiene conocimientos (por ejemplo: un menú de contexto).

En el último segmento del código que se muestra en la figura 10 se registra el nuevo componente mediante la función Ext.reg(). En Ext JS existen dos alternativas para instanciar componentes: una mediante el operador **new** del lenguaje javascript (en la figura 11 se muestra un ejemplo) y la otra consiste en un parámetro de configuración llamado xtype.

Ambas alternativas proporcionan un mismo resultado, pero existen ciertas diferencias que deben ser tenidas en cuenta. Cuando se usa el operador **new** el navegador crea la instancia

[7] Existen ciertos problemas que son necesarios tener en cuenta cuando se crean componentes usando este método, que no se presentan cuando son implementados usando el constructor. Por este motivo en este trabajo se decidió aplicar la vía del constructor, en aras de facilitar la creación de los componentes.

del componente en el momento en que inicia la aplicación. Esto podría resultar negativo en cuanto a rendimiento debido a que podrían ser instanciados componentes con los que el usuario nunca llega a interactuar, ocasionando un gasto innecesario de recursos del ordenador.

En cambio, cuando son instanciados usando xtype, son creados en el momento que el usuario los necesita, garantizando así que no sean instanciados innecesariamente. Por tanto para instanciar los componentes se utilizará de manera general **xtype** a no ser que algún requerimiento especial de tiempo de respuesta en algún caso de uso obligue al uso de **new**.

Hay que resaltar que los componentes instanciados mediante **new**, como han sido instanciados desde el comienzo de la aplicación, son mostrados más rápidamente pero la diferencia, en condiciones normales, no es relevante y el ojo humano no debe ser capaz de notarla.

Para poder crear los componentes mediante xtype primeramente deben ser registrados mediante la función Ext.reg(), la cual recibe dos parámetros: el primero es una cadena que representa el nombre nemotécnico por el que será buscado y creado el componente, y el segundo es el componente a registrar.

Un detalle de gran importancia a tener en cuenta cuando se desarrollan componentes, es el uso de la función Ext.ns() que implementa la librería. Esta función permite crear espacios de nombre dentro de javascript, algo que es de mucha utilidad para evitar colisiones en los nombres de los componentes, a la vez que mejora considerablemente la organización del código generado.

Implementación de la solución

Cuando se implemente la solución, deben seguirse estrictamente los estándares que se presentan a continuación, en aras de facilitar la reusabilidad y el mantenimiento del código desarrollado.

Componente por fichero

Cada vez que se desarrolle un componente, debe ser creado un fichero para el mismo. Esto es para facilitar la mantenibilidad al mismo. Y crear las bases para asociarle los metadatos necesarios para poblar el repositorio de componentes de manera tal que puedan ser usados eficazmente por todo los desarrolladores.

Nombres nemotécnicos

Comprender algo que fue desarrollado por otra persona puede resultar una tarea muy compleja cuando los nombres (de variables, clases, funciones, etc.) no son usados de manera que reflejen la función que desempeñan. Ese trabajo propone el uso de las siguientes convenciones cuando sean desarrollados los componentes:

- El nombre de las funciones debe comenzar con letra minúscula, y la primera letra de cada palabra subsecuente debe comenzar con mayúscula, con el resto de las letras en minúscula.
- El nombre de las clases o componentes debe comenzar con mayúscula, y la primera letra de cada palabra subsecuente debe comenzar con mayúscula, con el resto de las letras en minúscula.
- El nombre de las funciones o componentes debe reflejar la intención para la que fueron creados tal y como se especifica en el patrón ***IntentionRevelingName*** (Adolph, et al., 2001)
- Las variables deben ser declaradas en las primeras líneas de código y deben comenzar con letra minúscula, y la primera letra de cada palabra subsecuente debe comenzar con mayúscula, con el resto de las letras en minúscula.
- El nombre de las variables debe demostrar el propósito para el que fueron declaradas. De esta forma también se hace uso del patrón ***IntentionRevelingName***.

Comentarios

Durante el período de desarrollo el uso de comentarios es una necesidad, es decir, establecer las pautas que conlleven a lograr un código más legible y reutilizable, de manera que se pueda aumentar su mantenibilidad a lo largo del tiempo. Estos deben ser lo bastante claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando.

Antes de la declaración de una clase se escribe una breve descripción donde se explique el propósito de la misma. Que se escribe de la siguiente forma:

```
/**
* Nombre de la clase *
* Descripción *
* @autor *
* @package *(módulo)
* @version (versión - parche)
*/

/**Ejemplo*/

/**
* CapturarDatosContacto
*
* Componente para capturar datos de contacto
* @autor Mario Alberto Alvarez Garcia
* @package RRHH
* @subpackage Estructura y Composicion
* @version 1.0
*/
```

Figura 12 Ejemplo de metadatos para los componentes.

Identación del código

El código escrito debe estar correctamente identado en aras de mejoras la legibilidad del mismo, es por esto, que cada declaración de función, o cualquier estructura que utilice un bloque de código, situará el mismo una posición debajo, y la Identación a usar será un tabulador. A continuación se presenta en ejemplo que cumple con una correcta Identación

```
Ejemplo = Ext.extend(Ext.form.FieldSet, {
    title: 'Ejemplo de Identacion',
    defaultType: 'textfield',
    defaults: {
        anchor: '95%'
    },
    constructor: function(){
        this.items = [{
            xtype: 'combo',
            displayField: 'nombre',
            valueField: 'id',
            mode: 'remote',
            fieldLabel: 'Cargo',
            forceSelection: true,
            store: new Ext.data.JsonStore({
                root: 'cargos',
                fields: ['id', 'nombre']
            })
        }
        Ejemplo.superclass.constructor.call(this);
    }
});
```

Figura 13 Creación de un componente.

A continuación se presentara un ejemplo para demostrar cómo aplicar este estándar en la creación de componentes.

```
/**
* CapturarDatosContacto
* Componente para capturar datos de contacto
* de las estructuras de la AGR
*
* @autor Mario A. Alvarez Garcia
* @package RRHH
* @subpackage Estructura y Composición
* @versión 1.0
*/


Ext.ns('Rh', 'Rh.ec');
```

```
Rh.ec.CapturarDatosContanto = Ext.extend(Ext.form.FieldSet,{
  title:'Datos de contacto',
  layout:'column',
  autoHeight:true,
  defaults:{
    columnWidth:.5,
    layout:'form',
    defaultType:'textfield',
    labelWidth:80,
    defaults:{
      anchor:'95%',
      allowBlank:false
    }
  },
  constructor: function(){
    this.items = [{
      items: [{
        name: 'calle',
        fieldLabel: 'Calle'
      }, {
        name: 'entre',
        fieldLabel: 'Entre'
      }, {
        name: 'entreY',
        fieldLabel: 'Y'
      }, {
        name: 'edificio',
        fieldLabel: 'Edificio',
        allowBlank: true
      }]
    }, {
      items: [{
        name: 'numero',
        fieldLabel: 'Numero',
        allowBlank: true
      }, {
        name: 'apto',
        fieldLabel: 'Apartamento',
        allowBlank: false
      }, {
        xtype: 'combo',
        fieldLabel: 'Provincia',
        displayField: 'nombre',
        valueField: 'id',
        hiddenName:'provincia',
        mode: 'remote',
        forceSelection: true,
```

```
            triggerAction: 'all',
            emptyText: 'Seleccione...',
            readOnly: true,
            scope: this,
            store: new Ext.data.JsonStore({
               storeId: 'provinciasStore',
               url: 'estructura/provincias',
               root: 'provincias',
               fields: ['id', 'nombre']
            }),
            listeners: {
               'select': function(){
                  var munic = Ext.getCmp('municipioField');
                  if(munic.disabled)
                     munic.enable();
                  munic.clearValue();
                  Ext.StoreMgr.get('municipiosStore').reload({
                     params: {
                        idProvincia: this.getValue()
                     }
                  });
               }
            }
         }, {
            xtype: 'combo',
            id: 'municipioField',
            hiddenName: 'municipio',
            fieldLabel: 'Municipio',
            displayField: 'nombre',
            valueField: 'id',
            mode: 'local',
            disabled:true,
            forceSelection: true,
            triggerAction: 'all',
            emptyText: 'Seleccione...',
            readOnly: true,
            store: new Ext.data.JsonStore({
               storeId: 'municipiosStore',
               url: 'estructura/municipios',
               totalProperty: 'count',
               root: 'municipios',
               fields: ['id', 'nombre']
            })
         }]
      }];
      Rh.ec.CapturarDatosContanto.superclass.constructor.call(this);
   }
```

```
});
Ext.reg('ec:capturarDatosContacto', Rh.ec.CapturarDatosContanto);
```

El código precedente fue generado para capturar los datos de contacto (calle, número, edificio, municipio, provincia, etc.) que poseen las entidades aduanales que pertenecen a la AGR. Hasta el momento este componente es empleado para satisfacer los requerimientos de interfaz de un total de cinco casos de uso, que pertenecen al módulo de Estructura y Composición del Sistema Integral de Gestión de Recursos Humanos que es desarrollado en el PSTA.

El código del componente se encuentra almacenado en un fichero cuyo nombre es CapturarDatosContacto.js, es apreciable que este nombre sigue el estándar de nomenclatura definido en 3.2. De igual forma el nombre del componente deja claro el objetivo del mismo tal como se expresa en el patrón ***IntentionRevelingName***, para este caso **CapturarDatosContacto,** el nombre da la idea de que se presentaran la interfaz necesaria para que un usuario pueda escribir los datos asociados a un contacto determinado y que estos sean manejados por la aplicación y esto es exactamente el objetivo de este componente.

En las primeras líneas hay información disponible de los metadatos con una breve descripción del componente, su autor, el paquete al que pertenece y la versión del mismo. De esta manera fueron definidos en el estándar sobre la información que debía proveerse de los componentes desarrollados, aunque es válido aclarar que la idea final alrededor de este punto seria extender la información contenida en los metadatos.

La implementación del componente comienza con la declaración de un espacio de nombre, que es útil para mantener el código organizado y evitar colisiones de nombre con algún otro componente generado por un tercer desarrollador. Esto es realizado a través de la función Ext.ns().

Posteriormente comienza la implementación del componente CapturarDatosContacto, con la declaración del nombre del mismo y la asignación de la referencia que devuelve la función Ext.extend() a dicha variable, como fue explicado en el acápite anterior, correspondiente al diseño de la solución.

En el desarrollo de este componente no fue necesario redefinir ninguno de los métodos que forman parte del patrón template method que implementa el framework.

Validar la solución.

La idea que está presente a lo largo de todo este trabajo radica en la reutilización en la mayor medida posible del código ya implementado. Para ello es imprescindible garantizar que los componentes que se desarrollen cumplan a cabalidad con los parámetros de calidad que se definan de antemano.

El enfoque de validación que se propone no sigue los ya arcaicos métodos de validación centrados en probar el producto sino que siguen la propuesta de los sistemas de calidad total que establecen que el desarrollo de procedimientos de calidad llevará por su propio peso a obtener productos de calidad.

El elemento primario dentro de este enfoque estará centrado en comprobar que todos los desarrolladores de interfaz de usuario -dentro del polo llevan a cabo el procedimiento descrito en 3.2 para ello y de acuerdo a lo que plantea el modelo de calidad aprobado en la UCI, CMMI for Development versión 1.2 nivel 2 dentro del área de proceso PPQA donde pone de manifiesto que la adherencia a procesos es uno de los elemento claves.

El equipo de calidad deberá tener control de los requerimientos de presentación que han sido identificados en cada uno de los casos de uso descritos (los métodos y estrategias que deben ser utilizados para este fin están más allá del alcance de este trabajo), y debe chequear que para cada uno de ellos se haya realizado una entrada en la matriz de trazabilidad entre los requerimientos y el componente que le da respuesta al mismo. De igual forma se realizará el mapeo entre cada componente en esta matriz de trazabilidad y el fichero donde se encuentra implementado el referido componente.

Cuando se realiza esta operación además se hace necesario actualizar la tabla en la que se registra la frecuencia de aparición de cada requerimiento de información para asegurar que los componentes sean implementados de acuerdo a una correcta aproximación del algoritmo propuesto en este trabajo que se define en función de la frecuencia de aparición de los requerimientos.

El siguiente paso es comprobar que se siguieron los estándares de diseño y codificación establecidos, para ello el equipo de calidad debe confeccionar un grupo de listas de chequeos

que deben correrse para cada componente validando de esta forma que cada desarrollador ha realizado la implementación acorde al proceso definido en 3.2.

El último paso dentro de la etapa de validación está asociado a testear el funcionamiento del componente antes de que sea utilizado dentro de una aplicación específica. Para ello los miembros del equipo de calidad deberán confeccionar con la ayuda de la herramienta libre Selenium IDE, que permite realizar pruebas funcionales a elementos desarrollados con Ext JS, el set de casos de prueba que se le aplicara a cada componente registrando los resultados y presentando el listado de no conformidades al desarrollador para que este pueda corregirlos. Una vez más, los métodos que deben ser realizados para cumplimentar esta tarea, está más allá de los objetivos de este trabajo.

Documentar la solución

Esta tarea debe llevarse a cabo una vez que el componente ha sido testeado en su funcionamiento por el equipo de calidad y está aprobado para ser utilizado dentro de las aplicaciones del PSTA.

En este punto deben especificarse los elementos que componen los metadatos definidos previamente y confeccionarse una pequeña guía de uso del componente en la que se registraran los atributos básicos, las configuraciones básicas y la capacidad de extenderlas así como las funcionalidades que brinda el componente y la forma en que deben ser accedidas.

Esta documentación deberá ser asociada al fichero donde se encuentra la codificación del componente. Las plantillas necesarias para llevar a cabo las tareas de documentación no han sido consideradas como parte del alcance que pretende cumplimentar este trabajo.

CONCLUSIONES

A lo largo de este trabajo se abordó primeramente las situaciones que dieron lugar al mismo, las cuales consisten en el deficiente desempeño de los desarrolladores de interfaz de usuario del Sistema Único de Aduanas, originado por factores como escasa o ninguna reusabilidad del código y problemas en la capacitación en el uso de la framework, lo que trae consigo que se incumpla con los cronogramas de trabajo.

Se logro demostrar que aplicar las técnicas de desarrollo de software basada en componentes, a la capa de presentación del PSTA podría cambiar drásticamente la distribución de fuerzas y recursos necesarios para llevar a cabo estas tareas en los distintos proyectos.

Se logró proponer un procedimiento de desarrollo para la capa de interfaz que es capaz de guiar a los trabajadores de esta capa en dos momentos especifico, el actual donde no hay implementado ningún componente y que tiene como objetivo identificar cuáles son los componentes que deben ser implementados a la mayor brevedad sobre la base de utilizar algoritmos de asignación de prioridades del “primero el más frecuente”. En este sentido fueron identificados 10 componentes y se especifico su prioridad.

Fue definido y aplicado, en la codificación de los 5 primeros componentes, un procedimiento de diseño e implementación de componentes para el que se proponen estándares de codificación, patrones de diseño y mejores prácticas.

El segundo momento está asociado al estado deseable del proyecto con una alta cantidad de componentes desarrollados en este caso los desarrolladores solo tendrían que realizar una búsqueda en el repositorio de componentes hasta encontrar uno que se adapte a los requerimientos específicos o extender de uno de los existentes si se necesita alguna modificación.

Además de ello fueron establecidos los lineamientos básicos de documentación de cada componente que permita garantizar su uso en un entorno de colaboración.

RECOMENDACIONES

Durante el desarrollo de este trabajo se detectaron elementos que a juicio del autor pueden enriquecer la capacidad de respuesta de los desarrolladores en la capa de presentación pero que no entraban dentro del alcance del mismo y que son propuestos como líneas futuras de investigación y recomendaciones.

- Definir un estándar de nomenclatura para incluir metadatos a los componentes de interfaz
- Desarrollar una aplicación que permita manejar un repositorio de componentes de interfaz realizando búsquedas semánticas sobre ellos.
- Definir el conjunto de listas de chequeos y baterías de pruebas que permitan garantizar adherencia al proceso descrito y testear las funcionalidades del componente.
- Definir un conjunto de puntos de control dentro del procedimiento para realizar análisis de optimización como parte de un proceso de mejora continua.

BIBLIOGRAFIA

Adolph, Steve y Bramble, Paul. 2001. *Patterns for Effective Use Cases.* 2001.

Ariza, Maribel y Molina, Juan C. 2004. *INTRODUCCIÓN Y PRINCIPIOS BÁSICOS DEL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES.* 2004.

Bengtsson, PerOlof, y otros. 2000. *Analyzing Software Architectures for Modifiability.* 2000.

Bernaus, Albert y Blanco, Jaime. 2000. *Aprenda a crear paginas web. Curso de iniciacion.* 2000.

Cobo, Jose Antonio. 2008. *Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas.* 2008.

Eguiluz, Javier. 2007. *Introducción a AJAX.* 2007.

Eguíluz, Javier. 2007. *Introducción a Ajax.* s.l. : www.librosweb.es, 2007.

—. 2008. *Introduccion a Javascript.* 2008.

2009. Ext JS . *Ext JS - Client-side Javascript Framework.* [En línea] 2009. http://www.extjs.com/.

Guarachi, Franklin. 2005. *Reutilización de requerimientos.* 2005.

Gutiérrez, Javier J. 2005. *¿Qué es un framework web?* 2005.

Leavens, G.T. y Sitaraman, M. 2000. *Foundations of ComponentBased Systems.* s.l. : Cambridge University Press, 2000.

Lopez, Juan A. 2006. *Reutilización de software.* 2006.

Lynch, Daniel, Postel, Jon y Roberts, Lawrence. 2003. A Brief History of the Internet. [En línea] 2003. [Citado el: 5 de mayo de 2009.] http://www.isoc.org/internet/history/brief.shtml#Origins.

Mili, H, y otros. 1995. *Metamodeling in OO.* 1995.

Montero, Ramón. 2001. *XML Iniciacion y Referencia.* 2001.

Musciano, Chuck y Kennedy, Bill. 1999. *HTML, La guia completa.* s.l. : O'REILLY, 1999.

O'Reilly, Tim. 2005. What Is Web 2.0. [En línea] 9 de Septiembre de 2005. [Citado el: 6 de Mayo de 2009.] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

Powell, Thomas y Schneider, Fritz. 2004. *JavaScript 2.0: The Complete Reference, Second Edition.* s.l. : McGraw-Hill/Osborne, 2004.

Sperberg-McQueen, C. M. y Burnard, Lou. 1994. A Gentle Introduction to SGML. [En línea] 1994. [Citado el: 8 de Mayo de 2009.] http://www.isgmlug.org/sgmlhelp/g-index.htm.

Szyperski, Clemens A. 2002. *Component Software - Beyond Object-Oriented Programming – Second Edition.* s.l. : Addison-Wesley / ACM Pres, 2002.

The Dojo Toolkit. [En línea] http://www.dojotoolkit.org/.

Tretola, Rich. 2008. What is RIA? [En línea] 2008. http://www.insideria.com/2008/01/what-is-ria-1.html.

W3C, Ofinina Española. 2008. Guía Breve de CSS. [En línea] 2008. [Citado el: 4 de Mayo de 2009.] http://www.w3c.es.

Zammetti, Frank. 2007. *Practical Javascript, DOM Scripting and Ajax Projects.* s.l. : Apress, 2007.

ANEXOS

Anexo I ¿Qué clase extender?

No siempre es correcto extender de la clase Ext.Panel, ya que en ocasiones no son necesarias todas las funcionalidades que provee esta clase (cabecera, pie de página, barras de herramietas). Lo correcto es seleccionar la clase más baja en la jerarquía del framework que implemente las funcionalidades que se necesiten. A continuación se presenta un listado de que componentes son correctos extender, dependiendo de las necesidades funcionales:

- **Ext.Component** si solo es necesario un elemento HTML que encapsule algo que se va a ser mostrado y no necesita ser reajustado su tamaño por el contenedor al que pertenece.
- **Ext.BoxComponent** si el componente debe ser estar en un contenedor y es necesario que el mismo reajuste su tamaño.
- **Ext.Container** si el componente debe ser contenedor de otros componentes.
- **Ext.Panel** si son requeridas las características de los paneles.

Anexo II Ejemplos de componentes

A continuación se presenta la implementación de un componente que empleado para capturar los datos comunes de estructuras de la aduana (nombre y objetivos).

```
/**
* CapturarDatosComunesEstructuras
* Componente para capturar datos que son comunes
* a las estructuras de la aduana (nombre y objetivos)
*
* @autor Mario A. Alvarez Garcia
* @package RRHH
* @subpackage Estructura y Composición
* @versión 1.0
*/

Ext.ns(´Rh´, ´Rh.ec´);
Rh.ec.CapturarDatosComunesEstructuras = Ext.extend(Ext.form.FieldSet, {
  title: 'Datos Generales',
  autoHeight: true,
  layout: 'form',
  labelWidth: 80,
  defaults: {
    anchor: '97.5%'
  },
  constructor: function(){
    this.items = [{
      xtype: 'textfield',
      name: 'nombre',
      fieldLabel: 'Nombre',
      allowBlank: false
    }, {
      xtype: 'textarea',
      name: 'objetivos',
      fieldLabel: 'Objetivos',
      allowBlank: (this.options.tipoNodo == 'subUnidad') ? false : true
    }];
    Rh.ec.CapturarDatosComunesEstructuras.superclass. constructor.call(this);
  }
```

```
});
Ext.reg('ec:capturarDatosComunesEstructura, Rh.ec.CapturarDatosComunesEstructuras);
```

Anexo III Navegabilidad

Dado que el autor durante su investigación descubrió que los desarrolladores emplean diversos mecanismos cuando implementan la navegabilidad dentro de la aplicación, le pareció conveniente exponer aquí la solución óptima para la creación de este tipo de funcionalidad, la cual por sí misma no es suficiente para convertirse en un componente debido a que no es añadida funcionalidad alguna, sino que más bien se realiza empleando técnicas que implementa el framework y que se ha demostrado empíricamente que es desconocida por la mayoría de los desarrolladores.

Las diversas funcionalidades que implementa el sistema se exponen a través de módulos que integran los proyectos del PSTA (RRHH, API, LCF, etc.). Algunas de estas funcionalidades son:

- En el proyecto RRHH
  - El módulo de Estructura y Composición, permite crear nuevas estructuras como unidades, subunidades, grupos de trabajo, plazas, etc., a la vez que permite editar los datos de las mismas.
  - Selección al Ingreso, ofrece funciones como crear fichas, planificar fichas para la prueba, realizar análisis psicológicos
  - En el módulo de Trabajadores es posible insertar trabajadores, crear nuevos contratos, solicitar modificación de relación laboral
- En proyecto de API
  - El módulo Internet permite gestionar mensajes, generar reportes de información de vuelos, vuelos de pasajeros y provee información del estado de los mensajes y los mensajes basura

Para poder interactuar con el sistema, es necesario desarrollar un mecanismo que permita seleccionar la funcionalidad que el usuario desee.

Solución propuesta

Para corregir este problema, se propone que cada módulo debe poseer un componente con una barra de herramientas, en la cual se pueda acceder a las opciones que implementa el

mismo. Paralelamente, este componente actuará de contenedor de otros, en los cuales estarán implementadas las interfaces gráficas requeridas por cada funcionalidad.

Código de la funcionalidad

```
var componente = new Ext.Panel({
  id:'principal',
  layout:'card',
  tbar:[{
    text:'Boton 1',
    handler: function(){
      Ext.getCmp('principal').getLayout().setActiveItem('idFuncion1');
    }
  },{
    text:'Boton 2',
    handler: function(){
      Ext.getCmp('principal').getLayout().setActiveItem('idFuncion2');
    }
  }],
  items:[{
    xtype:'panel',
    id:'idFuncion1',
    title:'Funcion 1'
  },{
    xtype:'panel',
    id:'idFuncion2',
    title:'Funcion 2'
  }]
});
```

Del código precedente hay que resaltar lo siguiente:

- **<id: 'principal'>**: Debe estar presente en el componente, para posibilitar que el mismo sea referenciado a través de las funciones nativas del framework.
- **<layout: 'card'>**: Esta opción es vital en la propuesta de solución al presente problema. Un componente con **<layout:'card'>**, posibilita que solo uno de los componentes que contiene se visualicen a la vez. Esto quiere decir que un panel que contenga esta configuración, será capaz de contener un ilimitado grupo de otros componentes, los cuales se visualizan solo uno a la vez, dependiendo de la orden del usuario.

Anexo IV Estándar

Según la Organización de Estandarización Internacional (ISO)

- Un estándar o norma es un documento, establecido por consenso y aprobado por un organismo reconocido, que provee, para un uso repetido y rutinario, reglas, guías o características para las actividades o sus resultados, dirigidas a la consecución de un grado de óptimo orden en un contexto dado.

Las normas o pueden ser oficiales o "de facto"

- ***Norma oficial:***
  Es un documento público, elaborado por consenso, de acuerdo con un procedimiento establecido con el respaldo de un organismo reconocido.

En la clasificación de normas oficiales se distinguen:

- *Normas internacionales:* Poseen características similares a las normas regionales en cuanto a su elaboración, pero se distinguen de ellas en que tienen un alcance mundial. Las más representativas por su campo de actividad son las normas ISO elaboradas por la Organización Internacional de Normalización ISO u Organización Internacional de Estándares.
- *Normas nacionales*: Son elaboradas, sometidas a un período de información pública y sancionadas por un organismo reconocido legalmente para desarrollar actividades de normalización en un ámbito nacional.
- *Normas regionales:* Son elaboradas en el marco de un organismo de normalización de una región del mundo, normalmente de ámbito continental, que agrupa a un determinado número de organismos nacionales de normalización.

- ***Normas "de facto":***
  Normalmente impulsadas por fabricantes o grupos de interés que aunque no ofrecen las mismas garantías para el conjunto de las posibles partes interesadas que las normas oficiales, tampoco presentan la lentitud, el coste y la complejidad del proceso de las normas oficiales.

Beneficios que aportan los estándares.

Podría pensarse que la codificación basada en estándares es una tarea tediosa. A continuación se expone una lista con algunos de los beneficios que aportan los estándares de programación.

- ***Para los desarrolladores***
    1. Código fuente más comprensivo y fácil de mantener. Los programadores se familiarizan cada vez más con el estilo de codificación.
    2. Un enfoque uniforme para resolver los problemas resulta práctico. Los estándares de código revelan métodos recomendados que han sido empleados y probados en proyectos anteriores.
    3. Se necesita menos comunicación entre los desarrolladores y los administradores. Los desarrolladores no necesitan preguntar sobre los detalles de especificación del documento, ya que todo se encuentra documentado en los estándares de código.
    4. Es común para los programadores de menos experiencia reinventar la rueda. Cuando existen normas de codificación, hay una gran probabilidad de que todo problema no sea realmente un problema nuevo, sino que una solución fue documentada anteriormente.
- ***Para el aseguramiento de la calidad del equipo***
    5. Normas de codificación bien documentadas ayudan a la creación de scripts de prueba. Habiendo examinado el código fuente y probado la aplicación, basándose en el cumplimiento de las normas de codificación, se eleva considerablemente la garantía de calidad del software.
- ***Para los gestores del proyecto***
    6. Es importante para los gestores del proyecto mantener y garantizar la calidad del código fuente de sus proyectos. Implementar normas de codificación aumenta en gran medida que se logre este objetivo.
    7. Podrían evitarse repetidas dificultades en el rendimiento. Es común que el release de una aplicación sea menos impresionante en cuanto a rendimiento, cuando los datos reales de la nueva aplicación han sido cargado a la base de datos.

8. Menos consumo de horas-hombre, como resultado de sumar los esfuerzos en la aplicación de normas de codificación.
9. También es beneficioso para la organización que está aplicando la norma ISO 9001 de normas de codificación, porque es un complemento de la ejecución del plan de organización requisitos.

GLOSARIO DE TERMINOS

Software: Programas o elementos lógicos que hacen funcionar un ordenador o una red, o que se ejecutan en ellos, en contraposición con los componentes físicos del ordenador o la red.

POO (Programación orientada a objeto): La Programación Orientada a Objetos es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software, puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

MVC (Modelo Vista Controlador): Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web.

Javascript: Es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

AJAX (Asynchronous Javascript And XML): Es una técnica de desarrollo web para crear aplicaciones interactivas .Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

JSON (Javascript Object Notation): Notación de Objetos Javascript, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.