Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

DESARROLLO DE UNA APLICACIÓN WEB CONSTRUCTORA DE INTERFACES GRÁFICAS DE USUARIO PARA APLICACIONES DE ESCRITORIO

Trabajo Especial de Grado
Presentado ante la Ilustre
Universidad Central de Venezuela
Por los bachilleres

Madelyx Brianto Contreras
Javier Eduardo Sandoval Conti

Para optar por el título de Licenciado en Computación

Tutores:
Prof. Sergio Rivas
Prof. Joselito De Sousa

Caracas, Octubre 2007.

ACTA

Quienes suscriben, miembros del Jurado designado por el Consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los bachilleres Madelyx Brianto Contreras, CI: 16.869.543 y Javier Eduardo Sandoval Conti, CI: 15.758.153; con el título: **Desarrollo de una Aplicación Web Constructora de Interfaces Gráficas de Usuario para Aplicaciones de Escritorio**, a los fines de optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el día Viernes 26 de Octubre de 2007 a las 12:30 pm, para que sus autores lo defendieran en forma pública, lo que hicieron en la Sala 1 de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con la nota de _____ puntos.

En fé de lo cual se levanta la presente Acta, en Caracas a los veintiséis (26) días del mes de Octubre del año dos mil siete (2007) dejándose también constancia de que actuó como Coordinador del Jurado el Profesor tutor Sergio Rivas

_____	_____
Prof. Sergio Rivas (Tutor)	Prof Joselito De Sousa (Tutor)
_____	_____
Prof. Concettina Di Vasta (Jurado)	Prof. Andrés Sanoja (Jurado)

DEDICATORIAS

A Dios por bendecirme con el don de la vida, una familia hermosa, buenos amigos y muchas oportunidades…por darme la dicha de llegar hasta aquí y la inspiración para continuar.

A mis padres Max y Alix por su apoyo incondicional y por ir alumbrando siempre mi camino, siendo el pilar de mi existir y creadores de lo que soy. Por ser los mejores padres del universo. Los Amo.

A mi hermana Maylix por ser mí mejor amiga en todos los sentidos, confidente, compañera y cómplice en todas mis vivencias desde siempre y para siempre. Te Adoro.

A todas mis amigas y amigos con quienes compartí toda mi carrera y demás experiencias, por haber marchado conmigo hacia mi formación profesional y personal acompañándome en las buenas y en las malas. Los quiero.

Madelyx

AGRADECIMIENTOS

A nuestros tutores Sergio Rivas y Joselito De Sousa quienes con sus conocimientos, experiencia y asesoramiento constante fueron guía en la realización del presente documento, ayudándonos e inspirándonos a esforzarnos y dar lo mejor de nosotros como futuros profesionales. Muchas gracias por su tiempo y dedicación.

A la Universidad Central de Venezuela por brindarnos la oportunidad de formarnos profesionalmente y subir así un peldaño más en la larga escalera de nuestro desarrollo personal integral. Por Haber sido escenario Del logro de esta soñada meta de ser Licenciados en Computación.

A todos nuestros profesores, quienes con sus conocimientos nos proporcionaron las herramientas necesarias que hoy poseemos para salir al mercado laboral, y que fueron y son ejemplo a seguir.

A todas aquellas personas que de una u otra forma colaboraron con nosotros en este trabajo.

Madelyx Brianto y Javier Sandoval

RESUMEN

En los últimos años, en el área de la informática se ha considerado de gran importancia la interacción entre el humano y el computador, colocando al usuario como el elemento principal y factor fundamental en el proceso de evolución y orientación de la tecnológica.

El usuario principal de un sistema informático durante su proceso de creación es el desarrollador, quien debe enfrentarse a diferentes herramientas de desarrollo y manejarlas para cumplir con sus objetivos cumpliendo un conjunto de actividades. Es por eso que este trabajo de investigación consiste en la creación de una herramienta que facilite el diseño e implementación de interfaces gráficas de usuario para aplicaciones de escritorio, de fácil acceso y con los mínimos requerimientos para su uso. Dicha aplicación funcionará sobre plataforma Web para aprovechar las ventajas de las nuevas tecnologías que dota a las aplicaciones Web de interactividad, y dan al usuario el control del comportamiento de los elementos de la página; siendo capaz de cumplir con las tareas que actualmente realizan otras herramientas constructoras de interfaces de usuario existentes entre productos de software.

INDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

Desde siempre, durante la elaboración de un sistema informático los programadores se convierten en usuarios de una variedad de herramientas que dan soporte y facilitan sus actividades.

La implementación de una interfaz de usuario, bien elaborada, implica el uso de recursos y habilidades por parte de los desarrolladores tales como experiencia, creatividad, dedicación y tiempo, ya que generalmente las interfaces cuentan con muchos componentes que la hacen tediosa de implementar y/o modificar. Es por esto que algunos IDE (*Integrated Development Environment* - Entorno de Desarrollo Integrado) han incluido entre sus herramientas módulos que permitan el diseño y generación del código fuente de interfaces gráficas de manera sencilla e interactiva, donde el usuario pueda manipular los componentes de la interfaz y sus propiedades. Incluso se han desarrollado aplicaciones que son exclusivamente para la construcción de interfaces gráficas.

Actualmente, se pueden conseguir estas herramientas, pero para utilizarlas se presentan una serie de limitaciones por licencias, precios, requerimientos técnicos, entre otros que son obstáculos al momento del desarrollo. Por esto, sería una gran ventaja si dicha herramienta fuera fácilmente accesible, disponible para cualquier programador en el momento que lo necesite y desde cualquier lugar sin pasar por un proceso de obtención e instalación de un software en el computador en el que se encuentre. Estas características podemos encontrarlas fácilmente en una aplicación Web. Se podría pensar entonces en el desarrollo de una herramienta de generación de código fuente de interfaces gráficas de usuario para aplicaciones de escritorio en línea.

Con este Trabajo Especial de Grado se pretende realizar la investigación y aplicación de dos tipos de tecnologías en general: las que proveen dinamismo a las aplicaciones Web del lado del cliente, y las que soporten un esquema de transformación de contenido para poder lograr la generación del código fuente. El documento se organiza de la siguiente manera:

- En el primer capítulo se define el planteamiento del problema que se desea resolver, la justificación de la investigación y los objetivos a alcanzar durante la elaboración de la misma.

- El capítulo II comprende el marco teórico en el que se presenta la base informativa y conceptual de aquellos elementos que nos permitirán el desarrollo del proyecto: procesos de desarrollo de software a seguir, tecnologías del lado del cliente y tecnologías del lado del servidor a utilizar.

- En el capítulo III se presenta el marco aplicativo donde se explican y documentan todos los pasos realizados para lograr el desarrollo del sistema desde el análisis inicial, planificación, actividades, pruebas y resultados, hasta llegar a la versión final de la aplicación, siguiendo una adaptación del proceso de desarrollo XP.

- El capítulo IV consta de diferentes escenarios en los que los requerimientos de elaboración de interfaces gráficas de usuario fueron cubiertos utilizando la aplicación que es el producto de este trabajo.

- Finalmente se presentarán los resultados obtenidos de la investigación, las conclusiones a las que llegamos y las recomendaciones y mejoras para trabajos futuros relacionados.

CAPITULO I: PROPUESTA DE TRABAJO ESPECIAL DE GRADO

1.1 Definición del problema

Al momento de planificar el desarrollo de una aplicación de escritorio, la interfaz gráfica juega un papel muy importante y se puede convertir en una de las tareas más complejas de codificación en cualquier lenguaje de programación para configurar sus propiedades, apariencia y comportamiento; además toma una cantidad considerable de tiempo si no se cuenta con una herramienta que permita su diseño y generación de código de manera automática.

En muchos casos se debe conseguir e instalar una tercera aplicación que realice este trabajo o incluir un módulo especializado en creación de interfaces para lograrlo. Lo cual implica en ocasiones trabajar sólo con la versión de prueba de una de estas herramientas con límites de tiempo y/o funcionalidades si no es de licencia pública y el cumplimiento de requerimientos técnicos para su funcionamiento.

Finalmente, los resultados no siempre son los esperados tanto en diseño como en código fuente, lo cual complica el proceso de integración de la interfaz gráfica al resto del sistema, la expansión y mantenimiento del código.

1.2 Solución propuesta

Dado el problema anterior, se pensó en una nueva herramienta de desarrollo que simplifique las tareas y los posibles inconvenientes al momento de decidir implementar una interfaz gráfica para una aplicación de escritorio. Dicha herramienta sería provechosa al estar siempre disponible sin importar momento o lugar de implementación, sin limitaciones técnicas ni económicas para cualquier usuario; y al mismo tiempo facilitar el diseño de interfaces gráficas y generación de su código fuente.

Proponemos el desarrollo de una aplicación Web generadora de código fuente de interfaces gráficas de usuario (GUI: Graphical User Interface). El acceso a dicha aplicación para crear GUIs estaría abierto a cualquier usuario que lo desee y su fuente podría estar disponible para mejorarla y crear nuevas versiones con más funcionalidades útiles para los desarrolladores, inclusive creando interfaces para otro tipo de aplicaciones. La arquitectura de dicha aplicación se presenta en la figura 1, definiendo los módulos involucrados en un modelo de 3 capas.

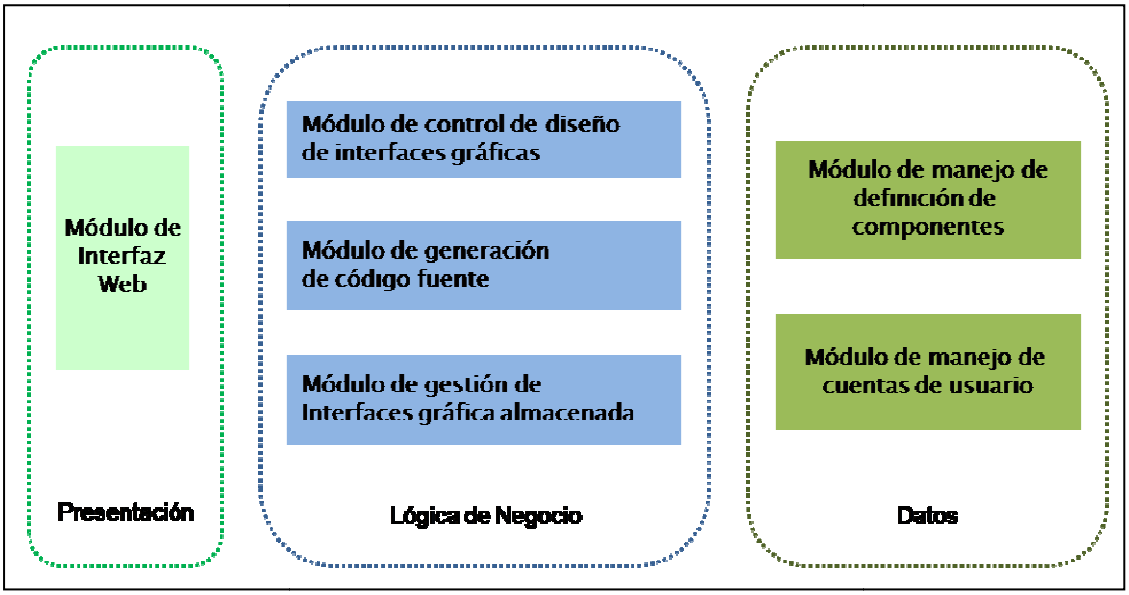


Figura 1: Arquitectura propuesta

La capa de presentación incluye el módulo relacionado a la interfaz del sistema constructora de interfaces gráficas a desarrollar que pondrá a disposición del usuario un área de diseño de GUIs para aplicaciones de escritorio y la opción de recuperarlas para editarlas o reutilizarlas. La lógica del negocio comprende los módulos encargados del procesamiento detrás del diseño y administración de los trabajos del usuario. Por último, la capa de datos cubre el manejo de datos persistentes del sistema asociados a los componentes implementados por el sistema y a las cuentas de usuarios con todos sus datos.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una aplicación Web que permita la construcción de interfaces gráficas de usuario para programas de escritorio y la generación del respectivo código fuente. Se tomará como caso de estudio las interfaces gráficas de usuario en lenguaje Java.

1.3.2 Objetivos específicos

- Aplicar Programación Extrema como proceso de desarrollo en conjunto con Modelación Ágil, para organizar la construcción de la aplicación Web planteada.

- Aplicar un mecanismo para manejar las diferentes propiedades de cada componente mientras se está diseñando la interfaz.
- Implementar un proceso que permita la creación de data XML con la información necesaria de la interfaz diseñada y realizar una transformación XSL para obtener el código fuente correspondiente.
- Desarrollar un proceso de manejo y descarga de archivos desde la aplicación.
- Diseñar e implementar un proceso que permita al usuario guardar y recuperar las interfaces realizadas con la aplicación al área de diseño para modificarlas y/o generar su código fuente nuevamente.

1.4 Definición de alcance y restricciones del proyecto

Luego de definir el problema y la solución propuesta, se establecieron las características y alcances del sistema de la siguiente manera:

- La versión de la aplicación será operativa en los browsers Mozilla Firefox v2.x, Opera Web browser v9.2x y Apple Safari 3.
- La aplicación será capaz de contemplar para el diseño de interfaces Java, con un subconjunto de componentes gráficos de la librería de Swing.
- Se podrá diseñar una ventana a la vez.
- Los eventos asociados a los componentes no son programables desde la aplicación.

CAPITULO II: MARCO TEORICO

Este capítulo presenta las bases conceptuales para la realización del sistema propuesto en el capítulo anterior, explicando el proceso de desarrollo a utilizar, describiendo brevemente las tecnologías Web involucradas en la implementación de una aplicación dinámica e interactiva y características de las interfaces de usuario de aplicaciones de escritorio y su codificación en lenguaje Java (Swing).

2.1 Procesos de Desarrollo de Software

El desarrollo de software no es una tarea fácil. Es por ello que existen varias propuestas que se enfocan en distintas aspectos del proceso de desarrollo. De un lado están las tradicionales que se centran especialmente en el control del proceso, estableciendo de manera estricta las actividades involucradas, los artefactos a producir, y las herramientas que se usarán. Estas propuestas aunque han demostrado ser efectivas y necesarias en algunos proyectos, han presentado problemas en otros, dando como resultado ser considerados procesos de desarrollo complejos que pueden incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto.

Por otro lado, están los procesos que se enfocan en el factor humano y producto de software. Esta es la filosofía de las metodologías ágiles, las cuales valoran al individuo (cliente y programador), la comunicación con el cliente y el desarrollo incremental del software en tiempos cortos; mostrando su efectividad en proyectos con requerimientos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [U- INSW].

Es por eso que debe seleccionarse un proceso de desarrollo que se adapte a las necesidades y objetivos de todos los involucrados. En esta oportunidad, nos encontramos frente a una propuesta de desarrollo con las siguientes características:

- La cantidad de desarrolladores es un grupo muy reducido (dos personas).
- El tiempo para el desarrollo es corto.
- Debe haber constante comunicación e interacción con el cliente (tutor).
- Su alcance puede ser variable.

Por lo tanto, hemos decidido seguir el camino de las metodologías ágiles y utilizar una combinación de Programación Extrema como proceso de desarrollo y prácticas de Modelación Ágil; y en este capítulo presentaremos el contenido de ambos, para recopilar detalles sobre los pasos a seguir para lograr su implementación.

2.1.1 Programación Extrema (Extreme Programming)[1]

Es un proceso de desarrollo de software que se basa en la reducción de la complejidad de desarrollo mediante un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales de los participantes en el proyecto, velocidad y eficiencia en el desarrollo. [U-PDD]

2.1.1.1 Objetivos

- Satisfacción del cliente: dar al cliente el software que necesite y cuando lo necesite.
- Potenciar al máximo el trabajo en grupo: Involucrar en el proceso de desarrollo a jefes de proyecto, clientes y desarrolladores.

2.1.1.2 Características

- Está orientada a quien produce y usa el software.
- Se basa en: simplicidad, comunicación, reutilización y mejora continua de código.
- Reduce el costo del cambio en todas las etapas de la construcción del sistema.
- Combina las mejores prácticas de software y las lleva al extremo.

2.1.1.3 Fases

A continuación se describen las cuatro (4) fases que define XP para el desarrollo de software y las prácticas a seguir que propone en cada una de ellas.

2.1.1.3.1 Planificación:

XP plantea la planificación como un permanente dialogo entre los participantes del proyecto para determinar:

- Ámbito: ¿Qué debe resolver el software para que genere valor al cliente?

[1] Extreme Programming: http://www.extremeprogramming.org/

- Prioridad: ¿Qué debe ser hecho en primer lugar?
- Composición de versiones: En cuanto el software aporte algo al negocio se deben tener listas las primeras versiones.
- Programación detallada: Dentro de una versión ¿Qué problemas se resolverán primero?
- Estimaciones: ¿Cuánto tiempo lleva implementar un requerimiento?
- Consecuencias: Informar sobre las consecuencias de la toma de decisiones por parte del negocio.
- Procesos: ¿Cómo se organiza el trabajo y el equipo? [U-FPE]

Actividades y prácticas:

Historias de usuario: El primer paso de cualquier proyecto que siga XP es definir las historias de usuario, estas constan de tres (3) ó cuatro (4) líneas escritas por el cliente en un lenguaje natural que representan sus requerimientos para el sistema.

Son usadas para estimar tiempos de desarrollo de la porción de la aplicación que describen. El tiempo de desarrollo ideal para una historia de usuario es entre una (1) y tres (3) semanas. Las historias de usuarios que requieran menos tiempo de implementación deben agruparse, mientras que aquéllas que necesiten más tiempo deben ser modificadas o divididas

Para establecerlas se toman en cuenta las siguientes prácticas:

- Deben ser entendibles por el cliente.
- Deben escribirse cada una por separado.
- Mientras más corta sea su descripción, será mejor.
- No deben hacer hincapié en los detalles, en algoritmos de implementación, ni en diseños de base de datos adecuados.
- Deben ser independientes unas de otras.
- Si la estimación duración de la implementación de una historia de usuario es de más de tres (3) semanas, se divide en un subconjunto de historias de usuario.
- Si se estima menos de una (1) semana, entonces se combina con una ó más historias de usuario.

Planificación de entregas: La planificación de un proyecto con XP incluye asignar valores a cuatro (4) variables: alcance, recursos, tiempo y calidad.

- Alcance: porcentaje de historias de usuario que pueden implementarse en el tiempo definido.
- Recursos: lo necesarios para el desarrollo del proyecto
- Tiempo: definir cantidad de tiempo para completar proyecto o parte de él
- Calidad: características deseables del software a desarrollar.

Una vez definidas las historias de usuario es necesario crear un plan de entregas, donde se indiquen:

- Las historias de usuario y sus prioridades que serán implementadas en cada versión del programa.
- Tiempos de implementación ideales de las historias de usuario
- Las fechas en las que se publicarán estas versiones.
- Los objetivos que se deben cumplir.
- El número de personas que trabajarán en el desarrollo
- Aspecto para evaluar la calidad del trabajo realizado.

Las reuniones periódicas serán una constante durante la fase de planeación, con todo el equipo para identificar problemas, proponer soluciones y señalar aquellos puntos a los que se les debe dar mayor importancia en un momento dado.

Iteraciones: El proyecto debe dividirse en iteraciones de aproximadamente 3 semanas de duración. Al comienzo de cada iteración el cliente debe seleccionar las historias de usuario definidas en la planificación de entregas que serán implementadas. También se seleccionan las historias de usuario que no fueron aprobadas por el cliente de la iteración anterior. Estas historias de usuario son divididas en tareas a desarrollarse entre 1 y 3 días de duración que se asignarán a los programadores. Cada iteración concluye ejecutando un conjunto de pruebas que permitan al cliente comprobar si está satisfecho con el resultado. En la figura 2 se puede apreciar la representación de este proceso.

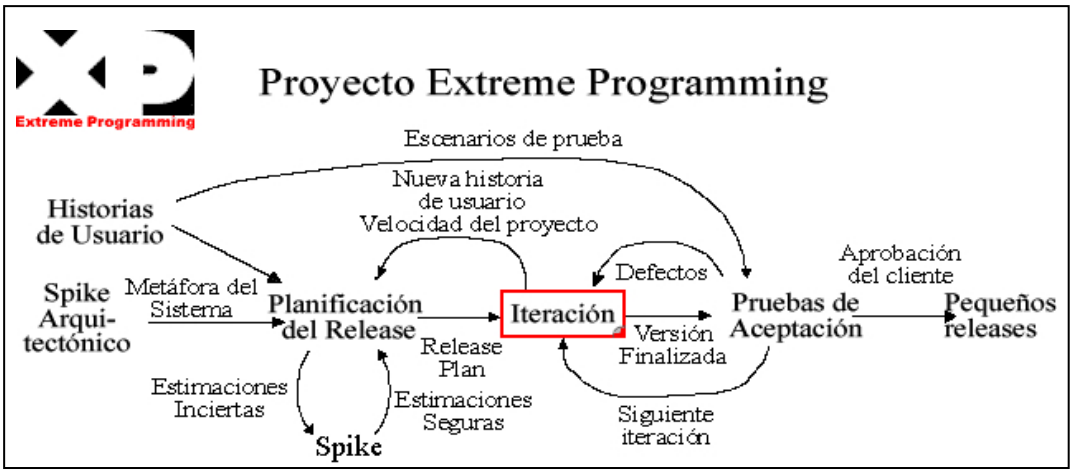


Figura 2: Esquema del proceso XP

Es importante que las entregas se realicen desde los primeros pasos de implementación y que con cada iteración el cliente reciba una nueva versión. Se aconsejan muchas entregas y muy frecuentes. De esta forma, un error en una parte esencial del sistema se encontrará pronto y, por tanto, se podrá arreglar antes. Además cada 3 a 5 iteraciones se deben revisar las historias de los usuarios y discutir nuevamente la planificación.

Velocidad del proyecto: La velocidad del proyecto es una medida que representa la rapidez con la que se desarrolla el proyecto. Se puede estimar de 2 formas:

- Cuántas historias pueden implementarse antes de una cierta fecha (planificación por alcance).
- Cuánto tiempo se requiere para desarrollar un conjunto de historias de usuario (planificación por tiempo).

De esta forma, se sabrá la cantidad de historias que se pueden desarrollar en las distintas iteraciones, y así controlar que todas las tareas se puedan desarrollar en el tiempo que dure la iteración.

En principio el tiempo se basa en una estimación, y al empezar a desarrollar las primeras historias de usuario se pueden cambiar los valores. Igualmente es conveniente evaluar esta medida cada 3 ó 4 iteraciones para determinar si el plan de entregas es adecuado y si no, modificarlo.

Para esta primera fase del proyecto XP propone las siguientes prácticas:

- ***Equipo completo:*** Forman parte del equipo todas las personas involucradas con el proyecto, incluido el cliente y el responsable del proyecto.
- ***Juego de la Planificación:*** Consiste en la construcción de un plan inicial para el desarrollo del sistema y a medida que éste avanza, se refina hasta que finaliza su implementación. XP propone la planificación como un juego, donde el cliente y los desarrolladores son los jugadores; las fichas, las historias de usuario, y los movimientos válidos son cada una de las responsabilidades de los jugadores. En principio la planificación se basa en determinar la velocidad del proyecto y estimaciones de tiempo de desarrollo de las diferentes historias de usuario, e intervienen 2 planificaciones importantes: planificación de iteraciones y planificación de entregas. Pero éstas se revisan frecuentemente a lo largo de todo el proceso de desarrollo, para poder realizar los cambios necesarios según sea conveniente para el proyecto.
- ***Cliente en el lugar:*** *S*e debe contar con el cliente durante todo el proceso de desarrollo para aclarar posibles dudas y poder tomar decisiones importantes que afecten al proyecto; y eliminar malentendidos internos que retrasen el desarrollo.
- ***Entregas Pequeñas:*** Es una práctica que implica la planificación y entrega de versiones del sistema que cumplan ciertas funcionalidades, manteniendo una constante retroalimentación por parte del cliente al utilizar, probar y analizar los resultados de la entrega realizada.
- ***Ritmo sostenible:*** Se debe trabajar a un ritmo que se pueda mantener indefinidamente. No debe haber días en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días.

2.1.1.3.2 Diseño

XP es un proceso ligero, esto es, que no pone demasiadas tareas organizativas sobre los desarrolladores (como modelado, generación de documentación, entre otras), cuyo efecto se minimiza gracias a la presencia del cliente.

El diseño crea una estructura que organiza la lógica del sistema, su correcta definición permite que los requerimientos crezcan con cambios controlados. Por lo tanto se plantea que el diseño debe ser revisado y mejorado de forma continua

según se van añadiendo funcionalidades al sistema, ya que a priori no se tiene toda la información suficiente para diseñarlo en su totalidad.

Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, hay que dividirla en varias. Si hay fallos en el diseño o malos diseños, estos deben de ser corregidos cuanto antes.

Para los diseños, sólo es necesario fichas, tarjetas o pizarras donde plantear las representaciones requeridas, ya que al implementar tareas pequeñas, no será necesario realizar inicialmente modelos complejos. Se propone invertir el tiempo en implementación y trabajar sobre el diseño antes de pasar a la próxima tarea.

XP propone aplicar las siguientes prácticas en el diseño:

- ***Diseños simples*:** XP sugiere construir diseños simples y sencillos que resulten fácilmente extensibles de implementar.
- ***Metáfora:*** El objetivo es mejorar la comunicación entre todos los integrantes del equipo al crear una visión global y común del sistema que se pretende desarrollar. La metáfora debe estar expresada en términos conocidos para los integrantes del grupo, por ejemplo comparando lo que se va a desarrollar con algo que se puede encontrar en la vida real. Aunque también se incluye información sobre los requerimientos, clases principales, relaciones entre ellas y patrones que se usarán en el sistema. Este aspecto se reforzará aplicando Modelación ágil, la cual se presentará más adelante.
- ***Riesgos*:** Si surgen problemas potenciales durante el diseño, XP sugiere utilizar una pareja de desarrolladores para que investiguen y reduzcan al máximo el riesgo que supone ese problema.
- ***Funcionalidad extra*:** No se debe añadir funcionalidad extra al programa aunque se piense que en un futuro será utilizada.

2.1.1.3.3 Codificación

El desarrollo es la pieza clave de todo el proceso de programación extrema. En todas las tareas se promueve el desarrollo a máxima velocidad, sin interrupciones y siempre en la dirección correcta; en XP se programará solo la funcionalidad que es requerida para la entrega en cuestión.

La presencia de la historia de usuario es necesaria a la hora de codificar. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que se hará y también tendrá que estar presente cuando se realicen las pruebas que verifiquen que la historia implementada cumple la funcionalidad especificada, dejando la optimización del código siempre para el final.

XP se concentra en la fase de codificación, promoviendo las siguientes prácticas como claves para el éxito del desarrollo:

- ***Programación en parejas:*** Sugiere la producción de código por parejas de desarrolladores, compartiendo una misma computadora. Con esta práctica se ha demostrado que dos programadores trabajando en conjunto generan, en el mismo tiempo, códigos de mejor calidad que trabajando por separado.
- ***Propiedad colectiva del código:*** Establece que todos los desarrolladores tienen permiso de realizar cambios en el código del proyecto para mejorarlo; y que la implementación de los diferentes módulos del sistema se rotará entre los equipos de desarrolladores para lograr una completa capacitación de todos en todo el producto y que se mantengan informados de las modificaciones realizadas en los módulos implementados.
- ***Integración Continua:*** Estipula que deberán realizarse integraciones del código del proyecto frecuentemente para evitar la mayoría de los problemas que surgen al integrar el trabajo realizado luego de mucho tiempo transcurrido. Esta práctica aumentará las posibilidades de detectar errores y corregirlos a tiempo.
- ***Refactorización:*** *E*s el proceso que consiste en modificar el código de una aplicación (o parte de esta) sin afectar su comportamiento. En XP los desarrolladores refactorizarán cuando sea necesario para obtener código más simple y legible. Con la refactorización se eliminan fuentes potenciales de errores y se disminuyen posibles problemas a otros desarrolladores.
- ***40 Horas a la Semana:*** *E*s el tiempo promedio de trabajo que debe cumplir el equipo de desarrollo del proyecto durante 5 días hábiles. De esta manera se logrará la utilización de recursos humanos a su máxima capacidad constantemente, sin afectar la calidad del producto final.
- ***Estándares de Código:*** La aplicación de estándares de código logra un ambiente de familiaridad con características comunes entre los archivos de código fuente creados por los desarrolladores. Además sin la aplicación de

éstos será difícil la refactorización, construcción de pruebas y la comprensión de códigos por parte de otros programadores.

2.1.1.3.4 Pruebas

*XP propone la división de las pruebas en 2 grupos: p*ruebas unitarias y pruebas de aceptación. Los desarrolladores realizan las pruebas unitarias a medida que se escriba el código. Y el cliente escribe las pruebas de aceptación para evaluar el cumplimiento de los requerimientos del sistema. Así, definiendo una serie de pruebas durante la construcción del proyecto, y aplicándolas cada cierto período de tiempo para validar los resultados obtenidos de las entregas, se asegura mantener la calidad esperada del producto final.

Las prácticas a aplicar en esta fase son las siguientes:

- Se deben crear las aplicaciones que realizarán las pruebas en un entorno de desarrollo específico para prueba.
- Hay que someter a pruebas las distintas clases del sistema omitiendo los métodos más triviales.
- Se deben crear los casos de prueba para evaluar y verificar procesamiento de los códigos antes de implementarlos.
- Las distintas pruebas se deben almacenar en el repositorio acompañado del código que verifican.
- El uso de las pruebas apoya la refactorización, ya que éstas permiten verificar que un cambio en la estructura de un código no tiene porqué cambiar su funcionamiento.

2.1.2 Modelación Agil (Agile Modeling)[2]

La modelación ágil es un proceso basado en prácticas para modelar eficazmente sistemas de software. Es una colección de prácticas dirigidas por principios y valores definidos para ser aplicados por los profesionales del software sobre un proyecto. No define un proceso completo de ingeniería de software; sino se concentra en las actividades de modelación y documentación. [U-AM]

2.1.2.1 Características

- No incluye actividades de programación, prueba, despliegue, operaciones o ayuda del sistema.
- Se enfoca en una porción del proceso total del software (ver figura 3) que se utiliza en conjunto con otro proceso bien definido que se adaptará a MA. Estos procesos cubren un alcance más amplio.
- *Es orientada a los usuarios y no a los procesos.* Explícitamente se busca que el desarrollo de software sea una actividad agradable.

2.1.2.2 Valores

La modelación ágil define cinco valores para lograr una comunicación y organización efectivas entre los integrantes del equipo de desarrollo. Estos son:

- **Comunicación.** Los modelos promueven la comunicación entre el equipo y los involucrados en el proyecto así como entre los desarrolladores del equipo.
- **Simplicidad.** Es mucho más fácil explorar una idea, y mejorarla a medida que aumenta la comprensión de la misma, dibujando un diagrama o dos en vez de la escritura de diez a centenares de líneas de código.
- **Retroalimentación.** El uso de diagramas y modelos permite analizar más fácilmente la evolución del proyecto y la corrección de errores en curso.
- **Coraje.** Es necesario tomar decisiones importantes y poder cambiar la dirección del desarrollo desechando o refactorizando el trabajo cuando hay algún error o se produce algún cambio.
- **Humildad.** Hay que tener presente que todos los desarrolladores, clientes y otros involucrados en el proyecto también tienen sus campos de especialización y pueden realizar aportes para el proyecto.

[2] Agile Modeling: http://www.agilemodeling.com/

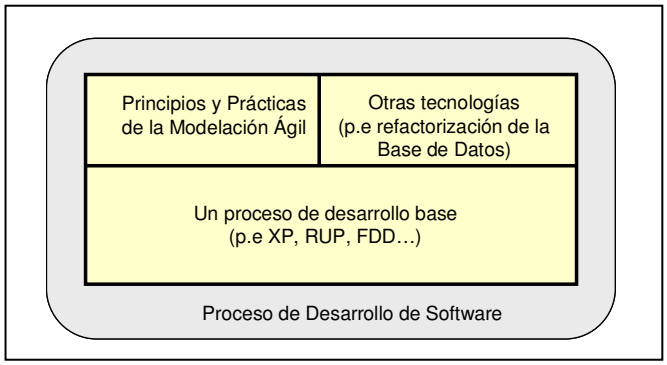


Figura 3: Esquema de ubicación de Modelación Ágil

2.1.2.3 Principios

Modelación ágil define principios que permiten formar la base para el desarrollo y cubrir las necesidades del ambiente de desarrollo.

- **Modelar con un propósito:** Claramente identificado desarrollarlo hasta el punto donde esté suficientemente exacto y detallado.
- **Maximizar la inversión de los involucrados:** Tiempo, dinero, instalaciones, entre otros para lograr desarrollar el software que cubra ciertas necesidades.
- **Marcas del recorrido:** Cada artefacto creado y considerado definitivo en un momento, se debe mantener almacenado por cierto tiempo para detectar impactos en caso de algún cambio en el proyecto o en otro documento.
- **Retroalimentación rápida.** El tiempo entre una acción y la retroalimentación con los resultados obtenidos en esa acción es muy importante.
- **Asumir la simplicidad.** Hay que mantener los modelos tan simples como sea posible, adaptarlos a los requerimientos actuales y no pensar en los futuros.
- **Aceptar el cambio.** Los requerimientos se definen en un cierto plazo, pero la comprensión de estos requerimientos pueden cambiar; además hay que tener presente que el equipo puede variar, tanto de personas como de puntos de vista.
- **Cambio incremental.** Es recomendable desarrollar un modelo que se ajuste a las necesidades del momento y modificarlo de una manera incremental a medida que el proyecto lo requiera.

2.1.3.4 Prácticas

La modelación ágil propone una serie de prácticas para el desarrollo fácil y eficiente de los proyecto.

Prácticas básicas:

- Participación activa de todos los involucrados en el proyecto.
- El modelo final debe representar los puntos de vista comunes de todos.
- Cambiar un artefacto cuando sea necesario.
- Probar con código cada modelo.
- Utilizar las herramientas más simples.
- Modelar en incrementos pequeños.
- Tener una sola fuente de información integrada.
- Todos los modelos y códigos deben estar disponibles para todo el equipo.
- Mantener el contenido de los modelos simples.
- Representar los modelos de forma simple.
- Publicar los modelos periódicamente para promover la comunicación abierta.
- Definir estándares a la hora de realizar modelos.
- Desechar los modelos temporales que ya no son útiles para el proyecto.
- Formalizar y decidir cuáles serán los modelos finales.
- Actualizar los modelos sólo cuando sea necesario.

2.1.3.5 Modelación Agil y Programación Extrema

En principio, es importante destacar que la creación de modelos forma parte del proceso de XP, las historias de usuario y las tarjetas CRC son ejemplos de esto. Por lo tanto podemos decir que modelación ágil tiene valores que aportar a XP, Sobre todo para resolver problemas internos que pueden surgir por la falta de documentos que registren puntos resultantes de las constantes conversaciones entre participantes del proyecto; la organización de los códigos y pruebas colectivas (que son documentos esenciales en XP) y dar formalidad a la definición de la arquitectura del sistema [U-AMXP].

UML se puede utilizar en proyectos de XP para cumplir con los requerimientos solicitados por el cliente y/o instrumentos de desarrollo y análisis del equipo. Un diagrama de la clase facilita la tarea de visualizar la relación entre las clases del sistema, y un diagrama de secuencia es muy buena representación de su comportamiento dinámico.

La idea es no complicar las tareas de programación. Si se necesita utilizar UML para diseñar un requerimiento que se está trabajando, es válido, pero no lo es pasar cantidades excesivas de tiempo perfeccionando diagramas del sistema

completo o funcionalidades extras que por lo general son sensibles a cambios continuos antes de lograr el producto final.

Otro punto que es importante destacar es la existencia de un nexo conceptual entre el enfoque, los valores y algunas de las prácticas de programación extrema con las de modelación ágil, sólo que el primero se enfoca en la implementación (código) y el segundo en el diseño (modelos): activa participación de los integrantes del proyecto, simplicidad de contenido, aplicación de estándares, propiedad colectiva, integración, crear estrictamente lo necesario para el momento, abrazar el cambio, realizar actividades en grupos y probar productos obtenidos; aplicándolas se mantiene el desarrollo del sistema sobre la misma línea guía para resultados efectivos.

2.2 Swing: Tecnología para la construcción de interfaces gráficas en Java

2.2.1 Interfaces gráficas de usuario

La interfaz gráfica es la parte del programa que permite la interacción con el usuario. Esta puede presentarse en diferentes estilos, que van desde la simple consola del sistema con las líneas de comandos hasta las complejas interfaces gráficas con barras de herramientas, menús, imágenes, tablas y otros que incluyen las aplicaciones modernas.

Una interfaz gráfica está construida en base a elementos gráficos básicos llamados componentes. Ejemplos típicos de estos componentes son los botones, barras de desplazamiento, etiquetas, listas, cuadros de selección y campos de texto. Los componentes permiten principalmente al usuario interactuar con la aplicación y proporcionarle información sobre las actividades y estado del programa.

El lenguaje Java incluye un conjunto de componentes para crear interfaces gráficas de usuario portables.

2.2.2 Swing

Desde sus inicios el entorno Java ya contaba con una librería de componentes gráficos conocida como AWT (Abstract Window Toolkit) que permitía utilizar los componentes propios de cada sistema operativo; pero en la práctica esta tecnología no funcionó. [U-SBG]

Al depender fuertemente de los componentes del sistema operativo, el programador que utiliza AWT solo dispone de las funcionalidades presentes en todos los sistemas operativos, porque están restringidos al denominador común de las características presentes en cada plataforma. Además, el comportamiento y apariencia de los componentes variaba entre sistemas, haciendo difícil la construcción de aplicaciones portables.

Por otra parte, se introducen los componentes de IFC (Internet Foundation Classes)[3] que eran mostrados y controlados directamente por código Java independiente de la plataforma (no requieren reservar recursos propios del sistema operativo). Además, al estar enteramente desarrollado en Java aumenta su portabilidad asegurando un comportamiento idéntico en diferentes plataformas. En 1997, Sun Microsystems y Netscape Communications Corporation anunciaron su intención de combinar IFC con AWT y crear una nueva librería para la creación de interfaces; así surge Swing.

Swing es un paquete gráfico compuesto por un amplio conjunto de componentes de interfaces de usuario que funcionan en el mayor número posible de plataformas. [U-SJFC]. Introduce un mecanismo que permitía que el aspecto de cada componente de una interfaz pudiese cambiar sin introducir cambios sustanciales en el código de la aplicación. Swing está totalmente escrito en Java emula la apariencia de los componentes propios del sistema manteniendo las ventajas de la independencia de la plataforma.

Entre sus características encontramos:

- **Amplia variedad de componentes**: en general las clases comienzan con "J" y son clases de este paquete que pueden añadirse a la aplicación para crear un componente de interfaz. Ejemplo: *JButton.*
- **Aspecto configurable (*look and feel*)**: se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto en algunos sistemas operativos (Metal Max, Basic Motif, Window Win32).
- **Arquitectura Modelo-Vista-Controlador**: esta arquitectura da lugar al enfoque orientado a objetos. Cada componente tiene asociado una clase de modelo de datos que lo describen, una interfaz que utiliza para su representación visual y un controlador para manipular su comportamiento y responder a eventos.
- **Objetos de acción**: cuando estos objetos están activados controlan las acciones de varios objetos componentes de la interfaz.
- **Contenedores anidados**: cualquier componente puede estar anidado dentro de otro.

[3] Internet Foundation Classes es una librería gráfica para el lenguaje Java desarrollada originalmente por Netscape y que se publicó en 1996.

- **Potentes manipuladores de texto**: además de campos y áreas de texto se presentan campos de sintaxis oculta para contraseñas y con múltiples fuentes.

2.2.2.1 Componentes y Contenedores

Cada elemento gráfico de una GUI es un componente y cada componente es una instancia de una clase del paquete swing que se crea y maneja como cualquier otro objeto Java.

Los componentes no se encuentran aislados, sino agrupados dentro de contenedores que se encargan de organizarlos y ubicarlos. Al mismo tiempo los contenedores son componentes y como tales pueden ser situados dentro de otros contenedores; y manejan el código necesario para el control del comportamiento de cada componente.

De las clases que conforman el paquete Swing, aquellas cuyo nombre comienza por J son propias de éste, las demás están incluidas en el paquete AWT. Las clases básicas al momento de construir una interfaz son:

- *Component*: clase abstracta que permite implementar cualquier componente que tenga representación gráfica.
- *Container*: componente que puede contener a todos los componentes gráficos.
- *JFrame*: clase que permite representar ventanas. Son contenedores, por lo cual incluyen panel de contenido (*Content Pane*) al cual se le pueden añadir componentes gráficos (botones, cajas de texto, etiquetas, entre otros) y otros contenedores como por ejemplo paneles (*JPanel*).
- *Layout*: controlan la forma en la que se distribuyen espacialmente los componentes dentro de un contenedor mediante los layouts managers, definiendo el diseño de la interfaz.
- *JPanel*: representa los paneles que pueden contener componentes gráficos y además tienen una superficie donde se puede dibujar (canvas o lienzo) para refrescar la visualización de un componente en pantalla cuando ésta varía.

La aplicación web generadora de interfaces gráficas de usuario a desarrollar, ofrecerá a sus usuarios los siguientes componentes gráficos para construir sus

diseños por ser éstos elementales y comunes en la mayoría de las aplicaciones de escritorio:

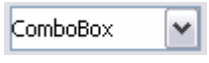
Componente	Clase	Vista gráfica	Descripción
Frame	JFrame		Ventana contenedora principal del resto de los componentes que conforman la interfaz.
Label	JLabel		Texto fijo que se coloca en la ventana.
Textbox	JTextComponent		Espacio para entrada de datos tipo texto (sólo una línea disponible).
Textarea	JTextArea		Espacio para ingresar texto (varias líneas disponibles).
RadioButton	JRadioButton		Componente que permite seleccionar una opción de un conjunto de opciones mutuamente excluyentes (sólo una a la vez de un grupo de opciones)
Checkbox	JCheckBox		Componente que permite seleccionar varias opciones de un grupo de valores.
ComboBox	JComboBox		Lista que se despliega o expande para presentar una lista valores del cual se puede seleccionar uno.
Button	JButton		Componente grafico que al ser pulsado ejecuta una o más acciones específicas.

Tabla 1: Componentes gráficos que manejará el generador s GUIs

2.2.2.2 Manejo de eventos

Las interfaces de usuario hechas en Swing son gestionadas por eventos, los cuales ocurren cuando el usuario interactúa con la interfaz.

Existen dos tipos básicos de eventos:

- **Físicos:** Corresponden a un evento hardware claramente identificable. Por ejemplo: Se ha pulsado una tecla.
- **Semánticos:** Se componen de un conjunto de eventos físicos, que sucedidos en un determinado orden tienen un significado más abstracto. Por ejemplo: EL usuario ha elegido un elemento de una lista desplegable.

El funcionamiento del modelo de eventos se basa en la gestión de excepciones. Para cada objeto que represente un componente de una interfaz gráfica, se pueden definir objetos oyentes (*Listener*), que esperen a que suceda un determinado evento sobre la interfaz para ejecutar una acción.

El gestor de oyente es un método que se invoca automáticamente como respuesta a un evento. Cada interfaz de oyente de eventos especifica uno o más métodos que deben ser definidos en la clase que implementa la componente de interfaz. Los objetos oyentes representan los controladores de Java.

Como todos los componentes Swing descienden de la clase *Component* de AWT, todo ellos soportan los siguientes eventos definidos en dicho paquete:

- *ComponentEvent*: Notifica a los oyentes cambios en el tamaño, posición o visibilidad del componente.
- *FocusEvent*: Notifica a los oyentes que el componente a ganado o perdido la posibilidad de recibir entrada desde el teclado.
- *KeyEvent*: Notifica a los oyentes las pulsaciones de teclas; sólo generado por el componente que tiene el foco del teclado.
- *MouseEvent*: Notifica a los oyentes las pulsaciones del mouse y los movimientos de entrada y salida del usuario en el área de dibujo del componente.
- *MouseMotionEvent*: Notifica a los oyentes cambios en la posición del cursor sobre el componente.

En la tabla #2 se pueden observar tipos de eventos, ejemplo de uno de los eventos de su categoría y su oyente asociado.

Tipo de Evento	Oyente de evento	Evento ejemplo
MouseEvent	MouseListener	OnClick
KeyEvent	KeyListener	OnKeyPressed
WindowEvent	WindowListener	windowClosed
FocusEvent	FocusListener	OnFocusGained

Tabla 2: Ejemplo de tipos de eventos y sus oyentes

Sin embargo, existen tipos de eventos como ActionEvent y ItemEvent que aunque están definidos en AWT sólo están disponibles para algunos componentes. Además Swing define nuevos tipos de eventos especiales para sus componentes nuevos, por lo tanto tampoco son soportados por todos los componentes de éste paquete. Ejemplos de ellos se presentan en la tabla 3.

Tipos de eventos definidos en Swing	Componente(s) a los que aplica
CaretEvent	JEditorPane, JPasswordField, JTextArea, JTextComponent, entre otros
ChangeEvent	JColorChooser, JButttton, JCheckBox, JMenuItem, entre otros
InternalFrameAdapter	JInternalFrame
ListSelectionEvent	JList, ListSelectionModel
UndoableEditEvent	JTextComponent

Tabla 3: Ejemplo de nuevos tipos de eventos definidos en el paquete Swing

Aunque todos los componentes Swing descienden de la clase *Container* del AWT, muchos de ellos no son usados como contenedores. Por eso, cualquier componente Swing puede generar eventos *Container*, que notifican a los oyentes que se ha añadido o eliminado un componente del contenedor. Sin embargo, en la práctica sólo los contenedores como los paneles, marcos, entre otros, generan eventos C*ontainer*.

2.2.3 Herramientas generadoras de interfaces gráficas Java (utilizando la librería Swing)

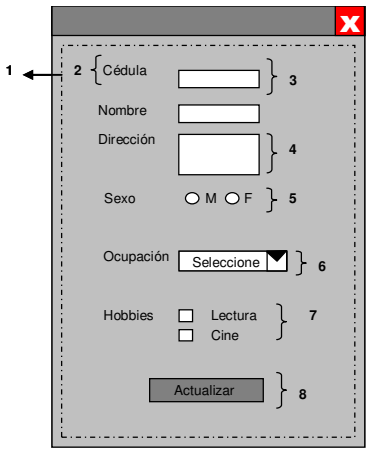
Con el fin de recordar las características, ventajas, desventajas de algunos de los programas que existentes capaces de construir y generar código fuente de GUIs para aplicaciones de escritorio, hemos recuperado notas importantes del análisis realizado en el documento: "*Estudio para el desarrollo de una aplicación Web generadora de código fuente Java de interfaces gráficas de usuario para aplicaciones de escritorio*" y las presentamos a continuación. [BMSJ2007]

Los programas constructores de interfaces gráficas de usuario tienen como propósito principal facilitar su creación, así en lugar de escribir el código fuente de la interfaz y luego ejecutarlo para ver la interfaz resultante, un programador puede diseñar la interfaz primero, ver el resultado y comenzar su programa a partir del código de la interfaz generada.

Se tomaron como muestra algunas herramientas existentes para la generación de interfaces gráficas de usuario en lenguaje Java. Éstas se nombran a continuación: JFrame Builder, NetBeans y SWT/Swing Designer.

2.2.3.1 Prototipo de interfaz a construir para las pruebas

Para poder medir las cualidades de las herramientas de manera equitativa, se elaboró con cada una de ellas la interfaz gráfica representada en la figura 4.



1 Panel
2 Etiqueta (Label)
3 Cuadros de texto (Text box)
4 Area de texto (Text Area)
5 Botones de opción (Radio button)
6 Lista desplegable (Select)
7 Cajas de selección (Checkbox)
8 Botón (Button)

Figura 4: Prototipo de interfaz a construir con cada uno de sus componentes

2.2.3.2 Criterios a considerar en el estudio comparativo de las herramientas

Se realizó una evaluación de cada una tomando en cuenta los criterios que se nombran a continuación. [BMSJ2007]

Usabilidad de la herramienta:

- ¿La interfaz es simple y sencilla visualmente?
- ¿Las funciones para la construcción de interfaces aparecen inicialmente?
- ¿Los controles de la herramienta para manipular los componentes están claramente visibles y sus funciones son identificables?
- ¿Minimiza las oportunidades para que el usuario cometa errores?

Funcionalidad de la herramienta

- ¿Se puede cambiar el tamaño de los componentes?
- ¿Se puede cambiar el color de los componentes?
- ¿Se puede cambiar las propiedades de los textos de la interfaz?
- ¿El código fuente asociado a la ventana se a generando en paralelo al diseño de ésta?

Características del código fuente generado

- ¿El archivo java generado incluye el encabezado con información sobre la clase?
- ¿El archivo java generado tiene documentación sobre los métodos y variables utilizadas?

Características de la interfaz generada

- ¿Aparecen todos los componentes utilizados en el diseño de la ventana?
- ¿La distribución de los componentes corresponde al diseño?
- ¿El tamaño de los componentes es el acorde a su contenido?
- ¿El color de los componentes es el especificado en el diseño?
- ¿El formato de la fuente es el especificado en el diseño?

2.2.3.3 Estudio de herramientas generadoras de interfaces gráficas en lenguaje Java

2.2.3.3.1 JFrame Builder 3.3.0

JFrameBuilder es una herramienta que permite la construcción de interfaces gráficas en lenguaje Java. Además ofrece la posibilidad de crear sofisticadas interfaces utilizando la técnica de "drag and drop", disminuyendo el tiempo de escritura de código Java; ya que éste se genera automáticamente en otra ventana de la herramienta. [U-JFB]. En su presentación menciona los siguientes atributos:

- Lo que se diseña es realmente lo que se consigue.
- Genera código en lenguaje Java legible.
- El código generado de Java no requiere ninguna librería adicional para compilar y funcionar.
- Abre archivos Java para recuperar el diseño de una interfaz.
- Maneja las características y eventos de los componentes con una interfaz comprensible y fácil de usar.
- Capacidad de fijar diferentes look and Feel de la interfaz.

Datos básicos de la herramienta:

- Herramienta de escritorio
- Distribución Shareware
- Disponible para Windows 98/ME/2000/XP/2003 Server com procesador 400MHz Pentium II.

La figura 5 muestra la vista de JFrame Builder con el prototipo de GUI construida, también se puede apreciar las partes de la sección de diseño y la presentación y ubicación de sus funcionalidades.

Por su parte, la figura 6 contiene una parte del código fuente generado por la herramienta en cuestión y el resultado de la interfaz al ejecutar dicho código.

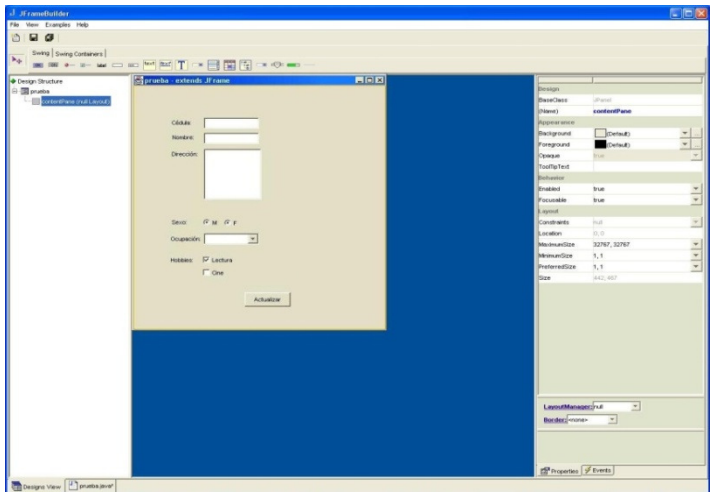
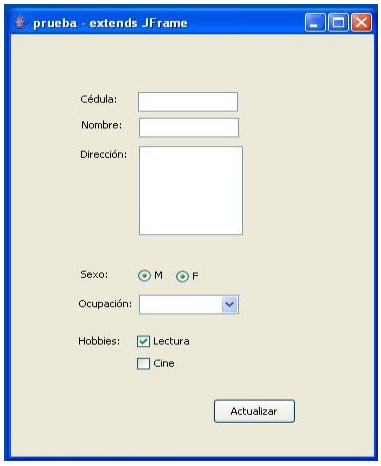


Figura 5: Vista de JFrame Builder con la interfaz construida




```
/**********************************************************************/
/*                    prueba                                          */
/*                                                                    */
/**********************************************************************/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
 * Summary description for prueba
 *
 */
public class prueba extends JFrame
{
        // Variables declaration
        private JButton jButton1;
        private JPanel contentPane;
        // End of variables declaration
        public prueba()
        {       super();
                initializeComponent();
                this.setVisible(true);
        }
        /**
         * This method is called from within the constructor to initialize the form.
         * WARNING: Do NOT modify this code. The content of this method is
always regenerated
         * by the Windows Form Designer. Otherwise, retrieving design might
not work properly.
         * to retrieve your design properly in future, before revising this method.
         */
        private void initializeComponent()
        {
                jButton1 = new JButton();
                contentPane = (JPanel)this.getContentPane();
                jButton1.setText("Actualizar");
                contentPane.setLayout(null);
                addComponent(contentPane, jButton1, 202,410,83,28);
                //
                // prueba
                //
                this.setTitle("prueba - extends JFrame");
                this.setLocation(new Point(0, 0));
                this.setSize(new Dimension(463, 509));
```

```
        }

        /** Add Component Without a Layout Manager (Absolute Positioning) */
        private void addComponent(Container container,Component c,int x,int
y,int width,int height)
        {
                c.setBounds(x,y,width,height);
                container.add(c);
        }
        //
        // TODO: Add any appropriate code in the following Event Handling
Methods
        //
        private void jButton1_actionPerformed(ActionEvent e)
        {
                System.out.println("\njButton1_actionPerformed(ActionEvent
e) called.");
                // TODO: Add any handling code here
        }
```

Figura 6: "Prueba.java": Resultado de la ejecución del código generado por JFrame Builder y extractos del código generado por JFrame Builder correspondiente a la definición de la ventana y al elemento Botón

Estudio de la herramienta:

Usabilidad de la herramienta	
¿La interfaz es simple y sencilla visualmente?	Si
¿Las funciones para la construcción de interfaces aparecen inicialmente?	Si
¿Los controles de la herramienta para manipular los componentes están claramente visibles y sus funciones son identificables?	Si
¿Minimiza las oportunidades para que el usuario cometa errores?	No
Funcionalidad de la herramienta	
¿Se puede cambiar el tamaño de los componentes?	Si
¿Se puede cambiar el color de los componentes?	Si
¿Se puede cambiar las propiedades de los textos de la interfaz?	No
¿El código fuente asociado a la ventana se a generando en paralelo al diseño de ésta?	Si
Características del código fuente generado	
¿El archivo java generado incluye el encabezado con información sobre la clase?	Si
¿El archivo java generado tiene documentación sobre los métodos y variables utilizadas?	Si
Características de la interfaz generada	
¿Aparecen todos los componentes utilizados en el diseño de la ventana?	Si
¿La distribución de los componentes corresponde al diseño?	Si
¿El tamaño de los componentes es el acorde a su contenido?	Si
¿El color de los componentes es el especificado en el diseño?	Si
¿El formato de la fuente es el especificado en el diseño?	Si

2.2.3.3.2 NetBeans 5.0

NetBeans es un entorno de desarrollo; una herramienta desarrollada por Sun Microsystem para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java. Es un producto libre y gratuito sin restricciones de uso [U-NB2]. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida. El módulo constructor de GUIs simplifica radicalmente el diseño de interfaces basadas en componentes Swing y AWT así como diseñar y configurar las propiedades, ubicación y tamaño de cada componente de la interfaz.

Datos de la herramienta:

- Herramienta de escritorio
- Distribución Freeware
- Disponible para plataforma Microsoft Windows 95, 98, 2000, NT con procesador P300 plataforma Solaris UltraSPARC 5: 333 MHz y plataforma Linux.

La figura 7 muestra la vista de NetBeans con el prototipo de GUI construida. Allí se aprecian las partes de la interfaz de la aplicación, de la sección de diseño y la presentación y ubicación de sus funcionalidades. La figura 8 contiene una parte del código fuente generado la herramienta en cuestión y el resultado de la interfaz al ejecutar dicho código.

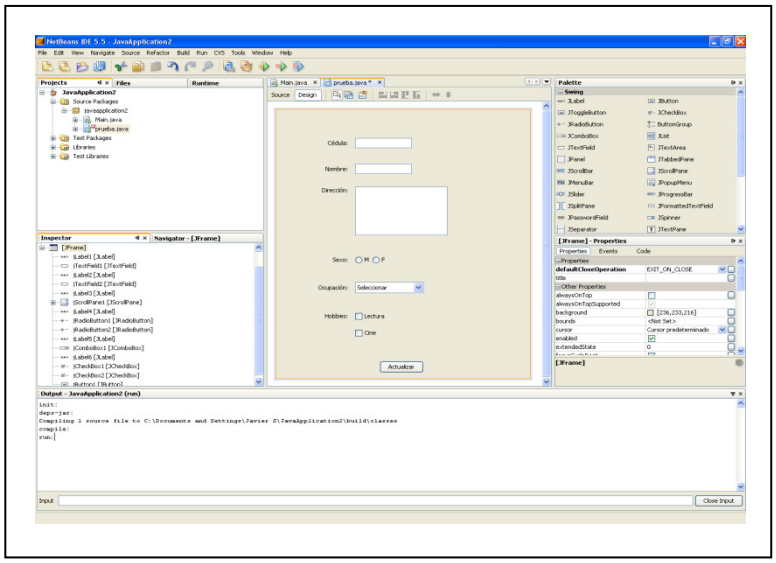
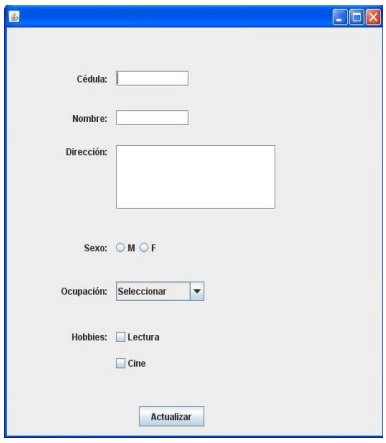


Figura 7: Vista de Netbeans con la interfaz construida




```
/*
 * Prueba.java
 *
 * Created on 28 de enero de 2007, 04:52 PM
 */
public class Prueba extends javax.swing.JFrame {

    /** Creates new form Prueba */
    public Prueba() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jButton1 = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jButton1.setText("jButton1");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(182, 182, 182)
                .addComponent(jButton1)
                .addContainerGap(211, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(275, Short.MAX_VALUE)
                .addComponent(jButton1)
                .addGap(77, 77, 77))
        );
        pack();
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new PruebaBoton().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    // End of variables declaration
}
```

Figura 8: “Prueba.java”: Resultado de la ejecución del código generado por NetBeans y extractos del código generado por NetBeans correspondiente a la definición de la ventana y al elemento Botón

Estudio de la herramienta:

Usabilidad de la herramienta	
¿La interfaz es simple y sencilla visualmente?	No
¿Las funciones para la construcción de interfaces aparecen inicialmente?	No
¿Los controles de la herramienta para manipular los componentes están claramente visibles y sus funciones son identificables?	Si
¿Minimiza las oportunidades para que el usuario cometa errores?	No
Funcionalidad de la herramienta	
¿Se puede cambiar el tamaño de los componentes?	Si
¿Se puede cambiar el color de los componentes?	Si
¿Se puede cambiar las propiedades de los textos de la interfaz?	Si
¿El código fuente asociado a la ventana se a generando en paralelo al diseño de ésta?	Si
Características del código fuente generado	
¿El archivo java generado incluye el encabezado con información sobre la clase?	Si
¿El archivo java generado tiene documentación sobre los métodos y variables utilizadas?	Si
Características de la interfaz generada	
¿Aparecen todos los componentes utilizados en el diseño de la ventana?	Si
¿La distribución de los componentes corresponde al diseño?	Si
¿El tamaño de los componentes es el acorde a su contenido?	Si
¿El color de los componentes es el especificado en el diseño?	Si
¿El formato de la fuente es el especificado en el diseño?	Si

2.2.3.3.3 SWT/SWING Designer 4.1.1 (plug-in para Eclipse)

SWT/Swing Designer es un plug-in o módulo diseñado para Eclipse que permite construir interfaces gráficas de manera sencilla. Esta herramienta se incorpora fácilmente para facilitar la construcción de GUIs en este programa. El plug-in tiene una vista de diseño y una de código, que genera el código fuente de la interfaz al momento que esta diseñando.

Datos básicos de la herramienta

- Tipo: Plug-in.
- Distribución Shareware.
- Disponible para Windows 98/ME/2000/XP, Linux +GTK2 (Cualquier versión de eclipse 2.0, 2.1, 3.0, 3.1 o 3.2)

La figuras 9 muestra la vista de Eclipse con el plug-in SWT/Swing Designer en uso y el prototipo de GUI construida, también se puede apreciar las partes de la sección de diseño y la presentación y ubicación de sus funcionalidades.

Finalmente la figura 10 contiene una parte del código fuente generado la herramienta en cuestión y el resultado de la interfaz al ejecutar dicho código.

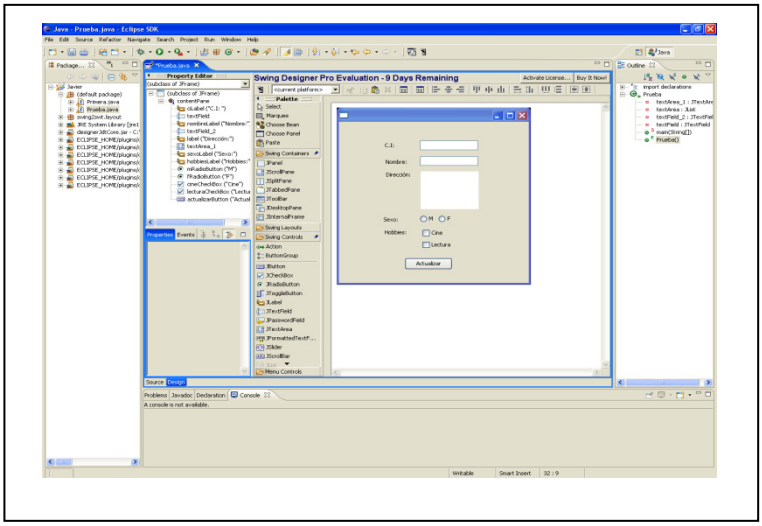
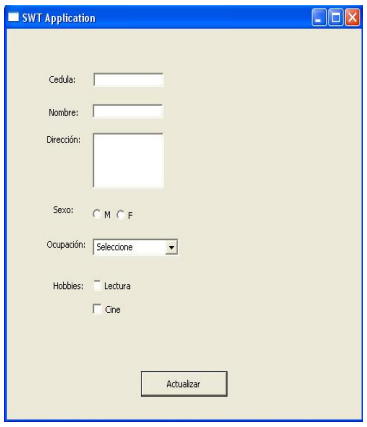


Figura 9: Vista de SWT/Swing Designer con la interfaz construida




```
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class prueba{

	protected Shell shell;

	public static void main(String[] args) {
		try {
			prueba  window = new prueba();
			window.open();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public void open() {
		final Display display = Display.getDefault();
		createContents();
		shell.open();
		shell.layout();
		while (!shell.isDisposed
	}
	protected void createContents() {
		shell = new Shell();
		shell.setSize(500, 375);
		shell.setText("SWT Application");

		final Button button = new Button(shell,
SWT.NONE);
		button.setText("button");
		button.setBounds(170, 245, 120, 30);
	}

}
```

Figura 10: “Prueba.java”: Resultado de la ejecución del código generado por SWT/Swing Designer y extractos del código generado por SWT/Swing Designer correspondiente a la definición de la ventana y al elemento Botón

Estudio de la herramienta:

Usabilidad de la herramienta	
¿La interfaz es simple y sencilla visualmente?	No
¿Las funciones para la construcción de interfaces aparecen inicialmente?	No
¿Los controles de la herramienta para manipular los componentes están claramente visibles y sus funciones son identificables?	Si
¿Minimiza las oportunidades para que el usuario cometa errores?	Si
Funcionalidad de la herramienta	
¿Se puede cambiar el tamaño de los componentes?	Si
¿Se puede cambiar el color de los componentes?	Si
¿Se puede cambiar las propiedades de los textos de la interfaz?	Si
¿El código fuente asociado a la ventana se a generando en paralelo al diseño de ésta?	Si
Características del código fuente generado	
¿El archivo java generado incluye el encabezado con información sobre la clase?	No
¿El archivo java generado tiene documentación sobre los métodos y variables utilizadas?	No
Características de la interfaz generada	
¿Aparecen todos los componentes utilizados en el diseño de la ventana?	Si
¿La distribución de los componentes corresponde al diseño?	Si
¿El tamaño de los componentes es el acorde a su contenido?	Si
¿El color de los componentes es el especificado en el diseño?	Si
¿El formato de la fuente es el especificado en el diseño?	Si

2.2.3.4 Análisis y comparación de las herramientas evaluadas

Como pudo observarse, ninguna de las herramientas cumple con todos los criterios que se consideraron; sin embargo, tomando en cuenta los resultados de cada evaluación, se pueden obtener las características comunes implementadas en la muestra de herramientas generadoras de código fuente Java de interfaces gráficas seleccionadas.

Es evidente que las herramientas más actuales y desarrolladas por grandes empresas productoras de software como Netbeans y SWT/Swing Designer, son más sofisticadas y tienen más funcionalidades, sin embargo suelen incluir una cantidad de elementos del proyecto que se está desarrollando que interfiere con la sencillez visual de la herramienta, es decir, hacen que la aplicación se vuelva muy compleja para manejarla. Pero también agregan más propiedades configurables a los componentes gráficos como: colores, fondo, formato de fuente, entre otras; que no aparecen en las herramientas más básicas como JFrame Builder; que si resultan muy útiles a la hora de diseñar.

Otro aspecto importante, es la manera de presentarle al usuario su área de trabajo. Esta característica solo la ofrece JFrame Designer, ya que es exclusiva para el diseño de interfaces, lo cual no ocurre con Netbeans puesto que es una aplicación más completa y se pueden realizar en ella otro tipo de aplicaciones. En Netbeans es necesario realizar una serie de pasos para crear un proyecto antes de llegar al área de trabajo. Y con el plug-in para eclipse es un caso parecido, ya que por estar incluido en Eclipse, se deben realizar un conjunto de pasos antes de comenzar a desarrollar (y si aún no está, debe instalarse previamente). Además es importante destacar que el proceso de instalación del plug-in es más complicada que una instalación común guiada por un asistente automático o wizard.

Por otro lado, la presentación, organización y documentación del código es muy importante porque está destinado a ser parte de otra aplicación y por lo tanto estará sujeto a modificaciones y a mantenimiento posterior. Todas de las herramientas cumplieron con esta característica, pero algunos con más detalles en su documentación (sobre las clases, información de los componentes, secciones del archivo, etc) que otros.

Por último se puede afirmar que uno de los aspectos más importantes que se debe mantener en una aplicación de esta categoría es la facilidad de acceso y con el 100% de sus funcionalidades, ya que se quiere que el usuario ahorre la mayor cantidad de tiempo y esfuerzo por escribir el código y obtener la interfaz gráfica sin problemas.

Al momento de diseñar el sistema propuesto se podrá extraer de éste análisis algunas de las ventajas y desventajas que se le presentan al usuario al momento de utilizar herramientas con las características que se mostraron anteriormente y así guiarnos para mantener en el sistema a desarrollar aquellos aspectos que promuevan la sencillez y eficacia en funcionamiento y resultados que se genere la aplicación.

2.3 Tecnologías de Desarrollo de Aplicaciones Web Dinámicas

Partiendo de HTML como el lenguaje descriptivo que da formato al contenido que se visualiza en un navegador, sabemos que con éste se pueden realizar tareas en una página Web de manera predeterminada, sin cambios en tiempo de ejecución; por lo que resulta bastante limitado.

Es por eso que ha sido necesario el empleo de otros lenguajes, capaces de responder a las peticiones del usuario en su actividad con el navegador. Así surgen las páginas Web dinámicas [U-PWDN]. El contenido de la página no está incluido en un archivo HTML ya que sus objetivos principales son mostrar el contenido partiendo de la información que solicita un usuario, generar páginas web de contenido estático y mejorar la interacción entre el usuario y el sitio web.

A continuación se presentan las propiedades de diferentes tecnologías que trabajando en conjunto permitirán construir una aplicación Web lo suficientemente interactiva como para lograr que sea el usuario el encargado de manipular los componentes que integrarán el diseño de interfaz gráfica y sus características, para crear un área de diseño de interfaces gráficas para aplicaciones de escritorio.

2.3.1 Tecnologías del lado del cliente

2.3.1.1 Cascading Style Sheets (CSS)

Las hojas de estilo o CSS son un mecanismo simple que utiliza una especificación de los estilos aplicables a un documento, describiendo cómo se verá en la pantalla [U-CSSHP].

Sabiendo que los estilos definen la forma de mostrar los elementos de un documento, CSS define dos tipos de ellos: el estilo lógico y el estilo físico. El estilo lógico se refiere a la composición del documento: cabeceras, párrafos, tablas, entre otros; sin tomar en cuenta la apariencia final, sino la estructura del documento. Por otro lado, el estilo físico si maneja la apariencia final (sin importar su estructura): párrafos con un cierto tipo de letra, tablas con un determinado color de fondo, los enlaces de diferentes colores o subrayados, y más.

La finalidad de las hojas de estilo es crear estilos físicos, separados de las etiquetas HTML (sin colocarlos como atributos de los elementos), y aplicarlos en los bloques de texto en los que se quieran aplicar.

2.3.1.1.1 Funcionamiento

CSS funciona a base de reglas que consisten en declaraciones que especifican el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas que se aplican a un documento HTML o XML. Una regla tiene dos partes: un selector y la declaración. A su vez la declaración se compone de una propiedad y el valor que se le asigne. Siguen la siguiente sintaxis:

- Regla: *selector {declaración}*
- Declaración: *propiedad:valor;*

Un ejemplo sería: h1 {color: red;}

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto a aplicar sobre el elemento. [U-GCSS]

El conjunto de reglas definidas en un CSS se pueden incluir en el código HTML de la página, escribirse en un archivo externo y hacer referencia a éste o importarlo desde una ubicación remota.

2.3.1.1.2 Atributos principales:

- Border: Define el formato del borde o espacio que limita un objeto, permitiendo cambiarle propiedades como el grosor, el tipo, el color, etc.
- Background: Permite modificar el fondo del objeto, bien sea con un cambio de color, o colocándole una imagen de fondo.
- Font: Permite configurar los estilos de letras, tamaño, estilo y color del texto contenido en un objeto.
- Cursor: Define el tipo de cursor que se va a mostrar cuando este se sitúe sobre un objeto.

2.3.1.2 Javascript

Javascript es un lenguaje de programación utilizado para crear pequeños programas o scripts encargados de realizar acciones dentro del ámbito de una página Web. Con Javascript se pueden crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador Web es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estas acciones.

2.3.1.2.1 Características

- Es un lenguaje basado en objetos y orientado a eventos, diseñado específicamente para el desarrollo de aplicaciones dentro del ámbito de Internet.
- Es sencillo y pensado para proveer rapidez y ligereza en las actividades realizadas sobre plataforma Web.
- Los programas en JavaScript pueden estar incluidos en los documentos HMTL y/o tener la referencia a un archivo externo de extensión 'js'; y se encargan de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos y manejar los elementos de la página.
- Es un lenguaje que permite tanto la programación de pequeños scripts, como de programas más grandes, orientados a objetos, con funciones y estructuras de datos complejas.

2.3.1.2.2 Eventos principales:

JavaScript permite controlar eventos que se producen en el sistema por la interacción del usuario. Por ejemplo, pulsar un botón es un evento, así como modificar el campo de un formulario o mover el ratón sobre un enlace. Se pueden definir manejadores de eventos que sean sensibles a éstos y actúen cuando ocurran. El resumen de los eventos principales y sus manejadores en el siguiente cuadro:

Evento	Se aplica a	Sucede si	Controlador del evento
Blur	Ventanas, frames y todos los campos de formularios	El usuario elimina el cursor del objeto	onBlur
Click	Botones, enlaces y formularios de confirmación	El usuario hace click sobre el elemento	onClick
Change	Campos de texto de formularios y listas de selección	El usuario modifica el valor del elemento	onChange
Error	Imágenes y ventanas	La carga de un documento o una imagen causa un error	onError
Focus	Ventanas, frames y todos los campos de formularios	El usuario sitúa el cursor sobre el elemento	onFocus
Load	Cuerpo de un documento HTML	El usuario carga la página correspondiente en el navegador	onLoad
mouseout	Áreas y enlaces	El ratón deja de estar encima del área o del enlace	onMouseOut

mouseover	Enlaces	El ratón está sobre el enlace	onMouseOver
Reset	Formularios	El usuario vacía un formulario con el botón de Reset	onReset
Select	Campos de texto de formularios	El usuario elige el campo del formulario	onSelect
Submit	Botones de envío	El usuario envía un formulario	onSubmit
Unload	Cuerpo de un documento HTML	El usuario descarga una página del navegador	onUnload

Tabla 4: Principales eventos en Javascritp

2.3.1.3 Document Object Model

Es una representación de los elementos de un documento estructurado, tal como una página web HTML o un documento XML, como objetos que tienen sus propios métodos y propiedades. Con DOM es posible acceder, añadir y/o modificar dinámicamente contenido estructurado dentro de una página Web. [U-WDOM]

Este modelo fue creado por W3C (World Wide Web Consortium), la organización internacional que define las normas, reglas y estándares para las tecnologías asociadas a Internet. DOM se creó como standard para la estructuración de documentos.

2.3.1.3.1 Funcionamiento

Cuando un navegador carga una página, crea una jerarquía de su contenido, lo cual representa la estructura del HTML (como se muestra en la figura 11). Esto genera una organización parecida a un árbol de nodos que pueden relacionarse, donde cada nodo representa un elemento, atributo o algún otro objeto.

Cada uno de estos objetos tendrá sus propios y únicos métodos y propiedades más un conjunto común de métodos y propiedades relacionadas con la estructura de árbol del documento para poder manipularlos.

DOM hace posible escribir aplicaciones que funcionen en todos los navegadores, servidores y plataformas; además de crear un modelo de documentos independiente al lenguaje de programación que se utilice para desarrollar.

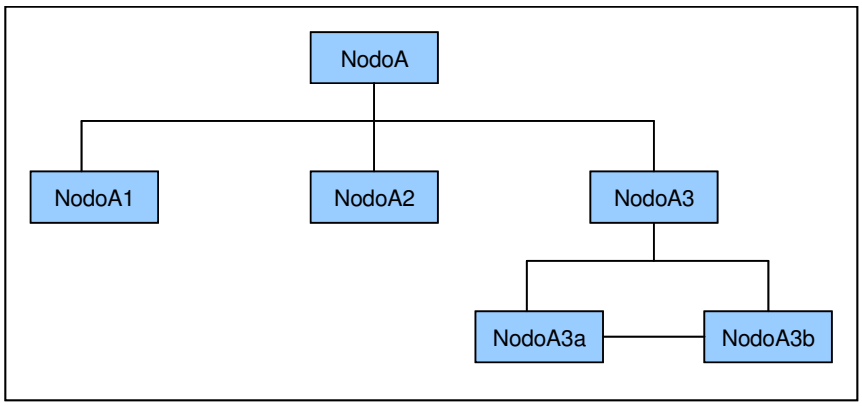


Figura 11: Árbol de estructura de objetos DOM

2.3.1.3.2 Funciones principales:

El siguiente cuadro describe las funciones principales, indicando nombre, parámetros, núcleo del DOM desde el cuál es accedida la función y su descripción:

Nombre	DOM núcleo	Descripción
getElementById (*Identificador*)	Método de *Document*	el nodo Elemento cuyo id es *Identificador*
getElementsByTagName (*Nombre*)	Método de *Document*	Devuelve una lista ordenada de todos los nodos Element que tengan como nombre de etiqueta *Nombre*
createElement (*Tipo*)	Método de *Document*	Crea un nodo tipo Element del tipo especificado en *Tipo*
createTextNode (*Cadena*)	Método de *Document*	Crea un nodo tipo Text con el valor *Cadena*
appendChild (*nuevoNodo*)	Método de *Node*	Añade el nodo *nuevoNodo* al final del conjunto de hijos del nodo al que se aplica.
insertBefore (*nuevoNodo*, *Nodo*)	Método de *Node*	Inserta el nodo *nuevoNodo* antes del nodo *Nodo* existente
cloneNode (*Recurrir*)	Método de *Node*	Devuleve un nodo libre, copia del nodo al que se aplica y con sus mismos atributos. Si *Recurrir* es verdadero se clonan también sus descencientes
nodeValue	Propiedad de *Node*	El valor del nodo. Dependiendo del tipo de nodo, esto tiene un sentido u otro (el valor del atributo en nodos Attr, el texto de los nodos Text, null en los nodos Element, etc.).
setAttribute (*Nombre*, *Valor*)	Método de *Element*	Añade al elemento un nuevo atributo *Nombre*, estableciendo *Valor*

Tabla 5: Principales funciones DOM

2.3.1.4 Dynamic HTML (DHTML)

DHTML es una nueva tecnología que permite crear páginas Web donde las actualizaciones del contenido sean constantes, se busque interacción del usuario con la página, se desee crear efectos y/o animaciones en las mismas, entre otras**.** [U-DHTML]

Con DHTML se consigue tener más control sobre la página, gracias a que los navegadores modernos incluyen una nueva estructura de visualización de contenido en páginas Web; denominada capa. Una capa es una división, una parte de la página, que tiene un comportamiento muy independiente dentro de la ventana del navegador ya que tiene sus propios parámetros: identificadores de posición, orden (por encima o debajo de otras capas), visibilidad y acciones asociadas a lo que suceda en el navegador o las actividades del usuario.

El la figura 12 se presenta mediante un ejemplo como se superponen diferentes elementos en una página Web con capas, pudiendo crear luego efectos sobre ellas como respuestas a ciertas acciones. Las capas pueden mostrarse o ocultarse en función de los parámetros que se elijan; si una capa se define como objeto móvil, puede desplazarse a voluntad por la pantalla, bien automáticamente o en respuesta a diferentes acciones, incluso el propio movimiento del ratón. Además se pueden ocultar, mostrar, desplazar y modificar.

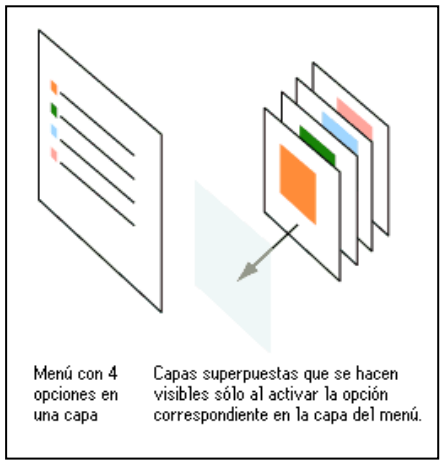


Figura 12: Representación de una página Web con un menú creado con capas

2.3.1.4.1 Características

- Convierte las etiquetas de HTML en objetos programables lo que permite manipularlos mejor.
- Todos los cambios de propiedades de los objetos se producen sin que tenga que volverse a cargar la página desde el servidor, y este no requiere una configuración especial.
- Se puede utilizar para crear animaciones y para introducir nuevas formas de navegación a través de los sitios Web sin sobrecargar el ancho de banda.
- Engloba las siguientes tecnologías: HTML ,CSS y JavaScript
- No requiere ningún tipo de plug-in para utilizarlo.

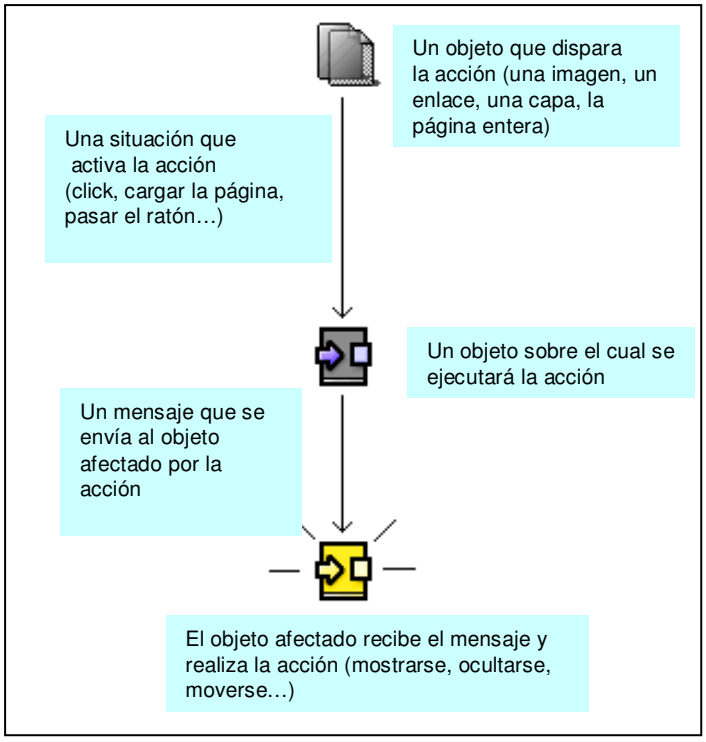


Figura 13: Elementos de una acción dinámica con DHTML

Es importante destacar la importancia de la combinación de las capas con CSS. Las capas son elementos HTML de tipo contenedor, que conforman en sí mismas zonas de la página independientes que se pueden crear, eliminar y posicionar de manera dinámica.

CSS permite la definición de estilo y presentación de elementos HTML, propiedades para el posicionamiento y control de elementos mediante capas.

Para el caso de la aplicación web a desarrollar, las capas podrán funcionar como los diferentes componentes de la interfaz que se diseñe. Y por tener atributos propios que la identifican unívocamente, sus propiedades visuales individuales podrán controlarse gracias a CSS y sus acciones como respuestas a diferentes eventos manejadas con Javascript.

2.3.1.5 Extensible Markup Language (XML)

XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados para usos específicos. Es una tecnología que ofrece una manera flexible de crear formatos de información comunes y compartir tanto los formatos como los datos entre aplicaciones informáticas, ya que la idea principal es estructurar los datos dentro de un documento, organizados siguiendo un formato específico para hacerlos legibles y claros para los humanos.

2.3.1.5.1 Ventajas

- Es independiente de la plataforma.
- Permite la organización de documentos de manera estructurada, facilitando así la definición de un modelo específico de procesamiento de la información.
- Un documento XML se puede definir utilizando nombres de etiquetas asignados o elegidos por la persona que esté desarrollando.
- Permite comunicar información entre sistemas de manera muy efectiva, ya que solo es necesario conocer el formato con el que están los datos.
- Es extensible, por lo tanto se pueden añadir etiquetas a la especificación del un lenguaje ya creado.

2.3.1.5.2 Estructura de un archivo XML

Document Type Definition (DTD): DTD es una tecnología que constituye un lenguaje propio para describir la estructura y sintaxis formal de un documento XML. Por lo tanto la validez de un documento XML se verifica con su DTD asociado. Este determina:

- Elementos: etiquetas permitidas y su contenido
- Estructura: orden en que van las etiquetas en el documento.
- Anidamiento: etiquetas que pueden ubicarse dentro de otras.

2.3.1.6 Asynchronous Javascript And XML (AJAX)[4]

AJAX no constituye una tecnología en sí, pero es un término que engloba a un grupo de éstas (presentadas anteriormente) que existen independientemente pero ahora trabajando en conjunto para desarrollar aplicaciones Web interactivas.

2.2.1.6.1 Características principales

- Hace peticiones al servidor sin tener que volver a cargar la página.
- Analiza y trabaja con documentos XML.

AJAX incorpora: presentación basada en estándares, exhibición e interacción dinámica, intercambio y manipulación de datos usando XML y recuperación de datos asincrónica. [U- AJAX]

2.2.1.6.2 Tecnologías que engloba AJAX

- HTML y CSS para presentar la información.
- DOM y JavaScript, para interactuar dinámicamente con los datos.
- XML para intercambiar y manipular datos de manera asíncrona con un servidor Web.
- Lenguaje de servidor que genera la información útil en XML y la envía al navegador.

Cuando se combinan estas tecnologías, las aplicaciones funcionan mucho más rápido, ya que se pueden actualizar las partes o porciones necesarias de la página Web y no cargarla completa de nuevo en el navegador.

2.2.1.6.3 Funcionamiento

Una aplicación AJAX cambia la naturaleza de la interacción en la Web introduciendo un intermediario (un motor AJAX) entre el usuario y el servidor. Sumar una capa a la aplicación podría pensarse que la haría menos reactiva, pero es lo contrario.

En lugar de cargar un página Web, al inicio de la sesión, el navegador carga al motor AJAX. Este motor es el responsable por generar la interfaz que el usuario ve y por comunicarse con el servidor en nombre del usuario.

[4]AJAX: http://www.ajax.org/

En la figura 14 se puede apreciar el modelo de actividad de AJAX en comparación al modelo de una aplicación Web tradicional, donde el motor en cuestión permite que la interacción del usuario con la aplicación suceda asincrónicamente (independientemente de la comunicación con el servidor). Así los lapsos de tiempo de respuesta del servidor a una petición serán imperceptibles para el usuario.

Cada acción de un usuario que normalmente generaría un requerimiento HTTP toma la forma de una petición al motor AJAX. Cualquier respuesta a una acción del usuario que no requiera un viaje de vuelta al servidor (como una simple validación de datos o edición de datos en memoria) será manejado por este motor. Ahora bien, si necesita algún recurso del servidor para responder (sea enviando datos para procesar, cargar código adicional, o recuperando nuevos datos) hace los pedidos asincrónicamente, usualmente usando XML, sin detener la interacción del usuario con la aplicación y enviando solo los datos necesarios para ahorrar tiempo y ancho de banda.

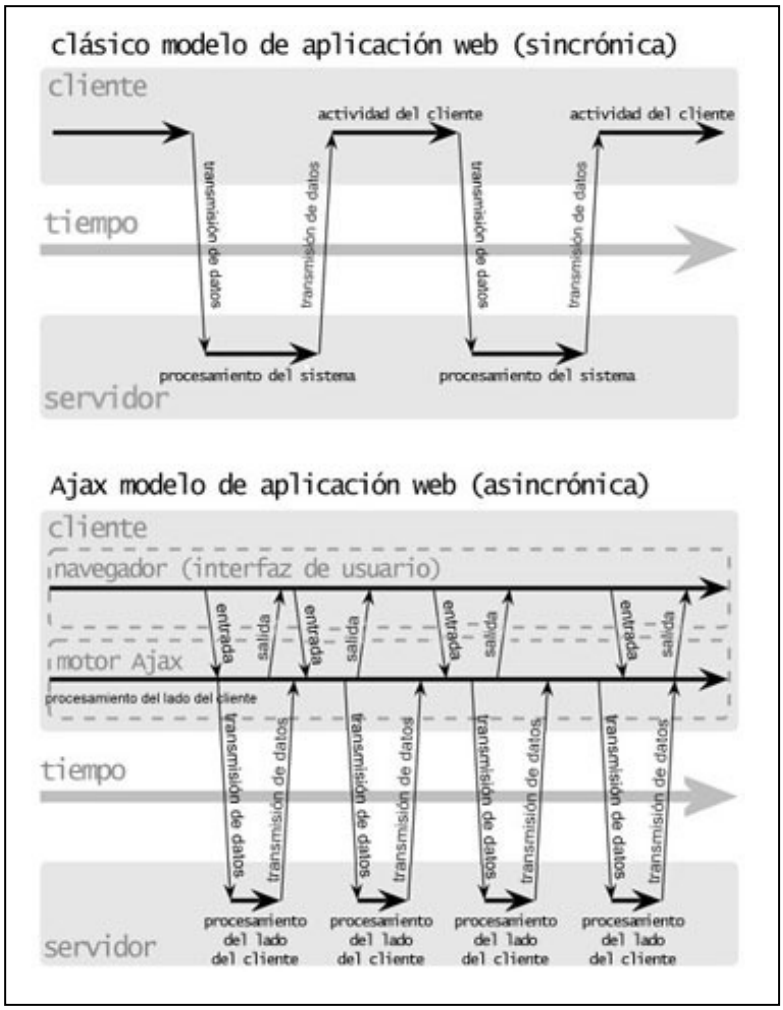


Figura 14: Comparación del funcionamiento de una Aplicación Web tradicional y Ajax

2.3.1.7 Prototype[5]

Prototype es un framework de distribución libre desarrollado en JavaScript por Sam Stephenson para el desarrollo sencillo de páginas Web dinámicas, el cual simplifica gran parte del trabajo cuando se pretende desarrollar páginas altamente interactivas. Su versión más actual es la 1.5 (recientemente terminada).

Está diseñado para extender las propiedades básicas del lenguaje JavaScript y regularizar las diferencias de implementación entre navegadores. Ofrece un estilo similar a otros frameworks orientados a objetos, pero posee la cualidad de ofrecer un extenso soporte de la tecnología AJAX, programación de más alto nivel (higher-order programming constructs) y una fácil manipulación del árbol DOM. [U-PROT]

[5]Prototype: http://www.prototypejs.org/

Su implementación resulta sencilla, se importa el archivo (librería) Javascript "prototype.js", y a partir de ese momento se tienen disponibles todas sus funciones y métodos que se especializan en el manejo de eventos y manipulación de los elementos de la página Web con nuevas funciones simplificadas para DOM.

2.3.1.7.1 Características

- Prototype aprovecha la flexibilidad de Javascript para enriquecer la funcionalidad de sus objetos principales (Object, Array, Number, String).
- Incluye nuevas clases utilitarias para trabajar con Ajax.
- Implementa un sencillo sistema de herencia para acercar a Javascript a las características de la programación orientada a objetos.

Prototype puede implementarse sobre las siguientes plataformas:

- Microsoft Internet Explorer para Windows a partir de la versión 6.0.
- Mozilla Firefox a partor de la versión Mozilla 1.7.
- Apple Safari a partir de la versión 1.2.

2.3.2 Tecnologías del lado del servidor

2.3.2.1 Patrón MVC

Es un patrón de diseño de arquitectura compuesto de tres módulos diferentes llamados: modelo, vista y controlador. El modelo está compuesto por el estado y datos que la aplicación representa, asumiendo la lógica del negocio que opera con los datos que se reciben o consultan de la fuente de datos del sistema. La vista es la interfaz de usuario que muestra información sobre el modelo en un formato específico y puede actuar como dispositivo de entrada para modificar dichos datos. Finalmente el controlador es el enlace entre los 2 elementos anteriores, haciendo corresponder las peticiones que llegan del cliente gracias a las actividades del usuario con las acciones correspondientes de la lógica del negocio y dirigir las respuestas a las vistas adecuadas y la navegación entre ellas.

2.3.2.2 Java Servlets

Los *servlets* son métodos de creación de páginas web dinámicas en el servidor usando el lenguaje Java. Son programas que funcionan atendiendo peticiones de un cliente teniendo al servidor como el encargado de realizar los procesos

necesarios y utilizando diferentes recursos para resolverlas. Están escritos en Java, con la ventaja de explotar todas las bondades de dicho lenguaje y su función es aquella que cumple el controlador en el patrón MVC.

Los servlets permiten desarrollar en Java y tener una abstracción orientada a objeto del protocolo http. Residen en contenedores Web estando fuertemente acoplados a demonios http y sirven peticiones de manera eficiente, ejecutándose en la máquina virtual de Java.

Al igual que los CGI, los servlets permiten que una petición sea procesada por un programa y genere una respuesta dinámica.

Los servlets adicionalmente definen un eficiente ciclo de vida que incluye la posibilidad de usar un único proceso para manejar todas las peticiones. Esto elimina la sobrecaraga de múltiples procesos de CGI y permite que el proceso principal comparta recursos entre múltiples Servlets y múltiples peticiones.

2.3.2.2.1 Caracteristicas

- Están escritos en Java y siguen un API bien estandarizado.
- El manejo de las peticiones se hace con hilos y no con procesos, reduciendo la carga para el procesamiento del servidor.
- Pueden comunicarse directamente con el servidor Web.
- Pueden comunicarse entre ellos.
- Se encuentran compilados en el contenedor Web, por lo cual su ejecución es muy rápida.

2.3.2.3 Extensible Stylesheet Language (XSL)

Es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio específico. [U-WikiXSL]

Esta familia está formada por tres lenguajes:

- XSLT (Extensible Stylesheet Language Transformations), lenguaje que permite convertir documentos XML de una sintaxis a otra.

- XSL-FO (XSL- Format Object), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF.
- XPath, o XML Path Language, es una sintaxis (no basada en XML) para acceder o referirse a porciones de un documento XML.

Los programas XSLT están escritos en XML, y generalmente, se necesita un procesador de hojas de estilo, o stylesheet processor para procesarlas, aplicándolas a un fichero XML.

Un mismo documento XML puede tener varias hojas de estilo XSL que lo muestren en diferentes formatos (HTML, PDF, RTF, VRML, PostScript, sonido, entre otros.). Además, la aplicación de una hoja de estilos XSL a un documento XML puede ocurrir tanto en el servidor o en el mismo navegador (algunos).

Una hoja de estilo XSL es una serie de reglas que determina cómo va a ocurrir la transformación. Cada regla se compone de un patrón (pattern) y una acción o plantilla (template). De este modo, cada regla afecta a uno o varios elementos del documento XML. El efecto de las reglas es recursivo, para que un elemento situado dentro de otro elemento pueda ser transformado también. Las hojas de estilo tiene una regla raíz que, además de ser procesada, llama a las reglas adecuadas para los elementos hijos.

El estilo de programación con las hojas XSLT es similar a lenguajes funcionales, tales como ML o Scheme. En la práctica, eso implica dos cosas importantes [U-GPWXX]:

- La programación es secuencial.
- No existen variables globales, ni ciclos (se emplea la recursión).

2.3.2.2.1 Resultado XSLT

El resultado de la transformación XSL de nuestro sistema será un archivo compuesto de líneas con instrucciones en código fuente Java para implementar una GUI, para lo cual es conveniente seguir estándares y buenas prácticas de programación.

Un estándar de programación es una forma de "normalizar" la programación de forma tal que al trabajar en un proyecto, cualquiera de las personas involucradas en el mismo comprenda el código fácilmente. La idea es definir la escritura y organización del código fuente de un programa dando mayor control y comprensión de éste para su mantenimiento, tomando en cuenta que casi nunca ningún software es mantenido durante toda su vida por su autor original. [U-CCEJ]

Por lo general los estándares de programación y las llamadas buenas prácticas definen la forma en que deben ser declaradas las variables y las clases, como deben escribirse los comentarios y en algunos casos que datos deben incluirse acerca del programador y versiones o cambios del código fuente, entre otros detalles. Todo esto dependiendo siempre del entorno y lenguaje de programación que utilices.

El en caso de Java, Sun Microsystems Inc. presentó en la especificación de dicho lenguaje, los estándares de codificación. A partir de ellos, seleccionamos las siguientes prácticas para aplicarlas al código a generar por el sistema generador de GUIs:

Un archivo fuente Java contiene una sola clase pública o un interface con el siguente contenido

1. Comentarios de inicio.
2. Sentencias Package e Import.
3. Declaraciones de clase e interface.

1.- Comentarios de Inicio: Debe incluir el nombre de la clase, y si se desea la información de versión, la fecha y las notas de *copyright*:

EJ:
```
/*
* Classname
* Version information
* Date
* Copyright notice
*/
```

2.- Sentencias Package e Import

La primera línea no comentada debe ser una sentencia package y luego pueden seguir sentencias import.

Ej:
```
package java.awt;
import java.awt.peer.CanvasPeer;
```

3.- Declaraciones de Clase e Interface: Deben aparecer los siguientes elementos en orden:

- Comentario de documentación de Clase/interface (/**...*/)
- Sentencia class o interface.
- Variables de clase (static).
- Resto de las variables.
- Constructores.
- Métodos: agrupados por funcionalidad para hacer la lectura del documento más fácil.

En todo el contenido se deben usar cuatro espacios como unidad de identación y evitar líneas mayores de 80 caracteres, ya que no son bien manejadas por muchos terminales y herramientas.

En cuanto a los comentarios los programas Java pueden tener dos tipos: de implementación y de documentación. Los comentarios de implementación están delimitados por /***...***/, y //. Los comentarios de documentación están delimitados por /****...***/.

Los comentarios de implementación se utilizan para explicar código o para hacer notas sobre la implementación en particular. Y los comentarios de documentación se usan para definir la especificación del código, desde una perspectiva libre de implementación para ser leído por desarrolladores que podrían no tener necesariamente a mano el código fuente.

Los comentarios deberían usarse para una introducción del código y proporcionar información adicional que no está disponible en el propio código. Sólo deberían tener información que sea relevante para leer y entender el programa.

Se recomienda una declaración de variable por línea, al principio de los bloques de código e inicializarlas donde son declaradas. La única razón para no inicializar una variable donde es declarada es si el valor inicial depende de algún cálculo que tiene que ocurrir antes.

Siempre se deberían usar dos líneas en blando en las siguientes circunstancias:

- Entre secciones de un archivo fuente
- Entre definiciones de clases e interfaces

Pero se debe usar una línea en blanco en las siguientes circunstancias:

- Entre métodos
- Entre las variables locales de un método y su primera sentencia
- Antes de un bloque de comentarios o un comentario simple
- Entre secciones lógicas dentro de un método para mejorar su lectura

En cuanto a nomenclatura los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas, intentando mantener los nombres de clases simples y descriptivos con palabras completas y evitando acrónimos y abreviaturas.

Ej: class Raster;

Los métodos deberían ser verbos, en mayúsculas y minúsculas con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas.

Ej: getBackGround();

Las variables, se escriben en mayúsculas y minusculas y con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas. Los nombres de variables no deben empezar con los caracteres subrayado "_" o dollar "$", incluso aunque estén permtidos.

Los nombres de variables deberían ser cortos y llenos de significado. La elección de una variable debería ser mnemónica-es decir, diseñada para indicar al observador casual su utilización. Se deben evitar los nombres de variable de un sólo caracter, excepto para variables temporales. Algunos nombres comunes de este tipo de variables son: i, j, k, m, y n para enteros.

Ej: float myWidth;
int i;

Los nombres de variables constantes de clases y las constantes ANSI deberían escribirse todo en mayúsculas con las palabras separadas por subrayados (“_”). (Se deberían evitar las constantes ANSI para facilitar la depuración.)

Ej: static final int MIN_WIDTH = 4;

2.4 Resumen del capítulo

En este punto de la investigación, al haber finalizado este capítulo se cuenta con las bases conceptuales necesarias para poder tomar de XP, las características, principios y prácticas vistas anteriormente y formar una versión que se pueda aplicar a las circunstancias de desarrollo y objetivos planteados al inicio de éste documento.

Igualmente se estudiaron un conjunto de tecnologías del lado del cliente desde el punto de partida con HTML, CSS y Javascript hasta llegar a frameworks actuales que han hecho evolucionar el concepto de dinamismo en aplicaciones Web; y tecnologías del lado del servidor para diferentes tareas: respuesta de peticiones del cliente, manejo de archivos XML y transformaciones de formato de data con XSL. Todas ellas con sus características y funcionalidades ofrecen la posibilidad de trabajar en conjunto para lograr desarrollar la aplicación que se plantea: interactiva y de contenido dinámico, donde los resultados de las acciones del usuario sobre la página serán dependientes del procesamiento de datos que viajarán del cliente al servidor con la ocurrencia de determinados eventos que indican la manipulación de los componentes de la página por el usuario.

CAPITULO III: MARCO APLICATIVO

En el marco aplicativo de este trabajo especial de grado se presenta paso a paso cómo se llevaron a la práctica de conjunto de conceptos, procesos y tecnologías y así lograr los objetivos planteados al inicio de esta investigación.

Inicialmente se presenta la descripción del proceso de desarrollo a seguir y su adaptación a este trabajo de investigación. Y seguidamente las etapas del proceso, actividades, resultados, artefactos y estrategias involucradas en el desarrollo de la aplicación.

3.1 Adaptación de XP para este trabajo

En esta oportunidad, el proceso de desarrollo guía fue XP en combinación con Modelación ágil lo cual nos permite la realización de la parte práctica de nuestro trabajo de investigación de manera rápida, sencilla, eficiente y lo suficientemente documentada. A continuación se describe como se aplican las 4 fases de la Programación Extrema al desarrollo de nuestro sistema:

- **Planificación:**

El primer paso fue tomar unos días para realizar un análisis global del sistema que se quería construir, creando una representación sencilla de las partes que lo conformarían y cómo se comunicarían entre ellas. Esto sigue la idea de XP de crear una metáfora del sistema que represente de forma general cuál es el resultado que se persigue en el desarrollo.

Además, se creó una lista de historias de usuario durante la conversación inicial con el cliente (tutor), quien fue responsable de ordenarlas para su implementación siguiendo una prioridad determinada por el aporte de la historia de usuario a la funcionalidad principal del sistema: generación de código fuente Java de GUIs. Luego la lista se dividió en dos grupos, uno con las historias de usuario relacionadas con el manejo de interfaces gráficas (prioritario en implementación) y otro relacionado a la gestión de usuarios en el sistema.

Se decidió trabajar en función del tiempo y así, cada historia de usuario tendría una cantidad de días estipulados (nunca mayor a tres (3) semanas o quince (15) días hábiles) para realizar actividades cortas que en conjunto dieran como resultado la implementación de la funcionalidad descrita en la historia de usuario en cuestión.

Igualmente, se establecieron 3 valores para medir el riesgo de implementación de cada historia de usuario (bajo, medio y alto) determinado subjetivamente por los desarrolladores basándose en su experiencia con las tecnologías a utilizar y el trabajo que implicaba las actividades a realizar. El formato de presentación de las historias de usuario se observa en la tabla 5.

Número:	*Nombre:*	
Prioridad:	*Riesgo en desarrollo:*	*Estimación del tiempo de implementación*
Descripción:		

Tabla 6: Formato seleccionado para manejar Historias de Usuario

Se realizó la planificación de iteraciones en las cuales se dividiría la implementación de las historias de usuario definidas, cada una con una duración de 3 a 4 semanas; y las entregas al cliente para realización de pruebas, correcciones y cambios se llevarían a cabo al término de cada iteración.

- **Diseño:**

Siguiendo las prácticas de Modelación Agil, para cada iteración en la etapa de diseño se agregaron los diagramas o modelos que se creyeron necesarios para el entendimiento y la documentación del sistema. Y en caso de haberlos creado en alguna iteración anterior, se actualizaron, desechando las versiones anteriores.

- **Codificación**

Antes de implementar se acordó entre los dos (2) programadores un estándar sencillo para codificar en lenguaje Java y Javascript:

- Seguir la nomenclatura, declaración de variables, espaciado, identación y documentación como se explicó en el Capítulo II en la sección 2.3.2.2.1 en la que se especifican los estándares a seguir en el código java generado por la aplicación (con excepción de los archivos Javascript que se decidió nombrarlos en minúscula).

- Con respecto a las sentencias de programación:
  - Cada línea debe contener como máximo una sentencia.
  - Una sentencia de retorno no debería usar paréntesis a menos que el valor de retorno sea más óbvio de esa forma.
  - Toda sentencia *switch* debería incluir un valor por defecto.

- No usar el operador de asignación en lugares donde pueda ser fácilmente confundible con el operador de igualdad.
- Manejar excepciones del lado del cliente y del servidor con un mensaje explicativo y descripción de la excepción.

Para la etapa de codificación de cada iteración se asignaban diferentes tareas por historias de usuario o historias de usuario completas a cada programador, éste al finalizar debía comunicar de inmediato a su compañero y acordar la integración del código y su almacenamiento al respaldo central a cargo de uno de ellos durante todo el desarrollo.

- **Pruebas**

Las pruebas unitarias se realizaron por historia de usuario, escribiendo los casos de prueba antes de comenzar a desarrollar (algunos de ellos con su código de verificación). Los casos no contemplados se agregaban al final de la implementación.

Las pruebas de las clases del lado del servidor se realizaron con JUnit[6] y las del lado del cliente se registraban en un archivo con el formato que se presenta en la tabla 6, con los detalles de sus resultados.

Las pruebas de aceptación se realizaron los días de las entregas con el cliente (tutor) quien siempre aportó ideas y mejoras para las versiones presentadas y las futuras a entregar.

Id Caso Prueba	Modulo	Descripción del Caso de Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla	Observaciones

Tabla 7: Formato de registro de pruebas unitarias del lado del cliente

- **Observaciones**

Al finalizar la iteración se utiliza una sección especial de observaciones si es necesario resaltar algún resultado, comentario o modificación que haya surgido durante las pruebas de aceptación con el cliente.

[6]JUnit: Conjunto de librerías utilizadas en programación para hacer tests unitarios de aplicaciones Java. http://www.junit.org/

3.2 Análisis Global del Sistema

Este análisis inicial parte de la propuesta realizada al inicio del documento y de la primera reunión con el cliente. De esta actividad se obtienen las historias de usuario y se construye un modelo del esquema de funcionamiento del sistema. Ambos resultados se presentan a continuación:

3.2.1 Historias de Usuario

Número: 1	*Nombre:* Elaborar interfaz de la aplicación web	
Prioridad: Alta	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 2 *días*
Descripción: Crear una página web con las secciones necesarias para proveer al usuario la información y las herramientas para trabajar con esta.		

Número: 2	*Nombre:* Registro de usuario	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 *día*
Descripción: El usuario debe registrarse en la página llenando un formulario con algunos datos personales y escoger un nombre de usuario y contraseña.		

Número: 3	*Nombre:* Recuperar contraseña	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 día
Descripción: La aplicación debe ofrecer la opción al usuario de recuperar su contraseña en caso que éste lo desee.		

Número: 4	*Nombre:* Autenticar usuario	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 día
Descripción: El usuario debe ingresar su nombre y contraseña para ingresar a la página y utilizarla.		

Número: 5	*Nombre:* Cambiar contraseña	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 día
Descripción: La aplicación debe ofrecerla la opción al usuario de cambiar su contraseña cuando lo desee.		

Número:6	*Nombre:* Barra de componentes gráficos.	
Prioridad: Alta	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 2 días
Descripción: Se le debe ofrecer al usuario de una paleta de componentes gráficos bien identificados que pueda utilizar en su diseño.		

Número: 7	*Nombre:* Área de edición de propiedades.	
Prioridad: Alta	*Riesgo en desarrollo:* Alto	*Estimación del tiempo de implementación:* 3 días
Descripción: Se le debe ofrecer al usuario un área donde se encuentren las características de un componente gráfico y se le permita editarlas.		

Número: 8	*Nombre:* Agregar componentes	
Prioridad: Alta	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 6 días
Descripción: El usuario podrá agregar componentes gráficos en el área de diseño.		

Número: 9	*Nombre:* Eliminar componentes.	
Prioridad: Media	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 2 días
Descripción: El usuario podrá eliminar componentes gráficos del área de diseño.		

Número: 10	*Nombre:* Editar componentes	
Prioridad: Alta	*Riesgo en desarrollo:* Alto	*Estimación del tiempo de implementación):* 10 días
Descripción: El usuario podrá editar los componentes gráficos y adaptarlo a sus necesidades		

Número: 11	*Nombre:* Generación del código fuente.	
Prioridad: Media	*Riesgo en desarrollo:* Alto	*Estimación del tiempo de implementación:* 9 días
Descripción: El sistema debe realizar el proceso de generación del código fuente de la interfaz gráfica cuando el usuario lo solicite		

Número: 12	*Nombre:* Obtención de archivo	
Prioridad: Media	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 3 días
Descripción: El cliente debe poder descargar el archivo .java asociado a la interfaz.		

Número:13	*Nombre:* Administrar interfaces almacenadas	
Prioridad: Media	*Riesgo en desarrollo:* Alto	*Estimación del tiempo de implementación:* 7 días
Descripción: El cliente debe poder guardar sus diseños y recuperarlos cuando desee para continuarlos y/o generarlos de nuevo.		

Del grupo anterior se seleccionaron como historias de usuario con mayor prioridad aquellas relacionadas a la construcción de interfaces de usuario (producto del sistema). Estas se nombran a continuación:

HU 1.- Elaborar interfaz de la aplicación web.

HU 6.- Barra de componentes gráficos.

HU 7.- Área de edición de propiedades.

HU 8.- Agregar componentes.

HU 9.- Eliminar componentes.

HU 10.- Editar componente.

HU 11.- Generación de código fuente.

HU 12.- Obtención de archivo.

HU 13.- Administrar interfaces almacenadas.

Las historias de usuario restantes fueron aquellas relacionadas con el manejo de los usuarios en el sistema y se implementarían después:

HU 2.- Registro de usuario.

HU 3.- Recuperar contraseña.

HU 4.- Autenticar usuario.

HU 5.- Cambiar contraseña.

3.2.2 Metáfora (Estructura del sistema)

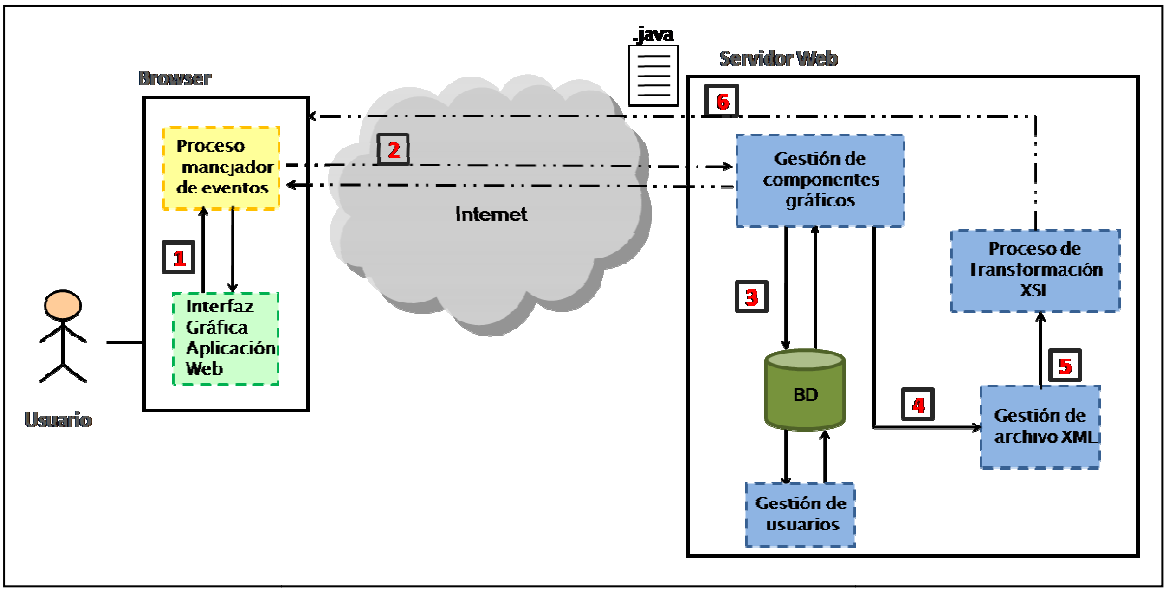


Figura 15: Metáfora (Estructura del sistema)

- **Interfaz gráfica de la aplicación Web:**

Sitio Web al servicio de los usuarios que se compone de un ambiente de diseño sencillo con un área de trabajo adecuada y un conjunto de herramientas y funcionalidades organizadas para la manipulación de componentes gráficos dentro de la página Web.

- **Proceso manejador de eventos:**

Encargado de controlar las diferentes acciones a ejecutar dependiendo de las actividades del usuario sobre los componentes de la interfaz que está diseñando en el área de trabajo del sistema Web. Hace posible el dinamismo de la página y la actualización de las propiedades y características de los componentes gráficos al instante que el usuario realiza modificaciones sobre éstas.

- **Gestión de componentes gráficos**

Proceso encargado de proveer los datos de los diferentes tipos de componentes disponibles a utilizar en el diseño de GUIs en el sistema. Además responsable del almacenamiento de los atributos asociados a los componentes gráficos que el usuario crea y manipula durante la construcción de sus diseños en la página. Estará en constante actividad para mantener la información de cada componente actualizada luego de cada cambio que el usuario realice sobre ellos.

- **Gestión de archivo XML:**

Encargado de reunir los datos necesarios de los componentes que conforman la interfaz diseñada en el área de trabajo de la página Web, estructurarlos en un formato XML establecido y crear un archivo físico que se mantendrá almacenado en la cuenta del usuario para su uso posterior.

- **Proceso de transformación XSL:**

Encargado del procesamiento del archivo XML, aplicándole una transformación XSL para acceder a los datos contenidos en él, darle formato de código fuente java según la información que represente y generar la salida en un nuevo archivo de extensión ".java".

- **Gestión de usuarios:**

Administración de cuentas de usuarios registrados incluyendo las diferentes interfaces realizadas con la aplicación que serán almacenadas para ponerlas a disposición del usuario cuando lo desee.

3.2.3 Especificaciones técnicas

El desarrollo del sistema se implantará utilizando las siguientes herramientas, tecnologías:

- Servidor Web : Apache Tomcat 5.5.9.
- Procesamiento de Eventos: JavaScript 1.3 y Prototype 1.5.
- Intercambio de información cliente-servidor: AJAX.
- Proceso de transformación XSL: javax.xml.transform (jdk1.6.0_02).
- Gestión de archivos Java: java.io (jdk1.6.0_02).
- Gestión de usuarios y sus trabajos: SMBD MySQL 5.0 .

Una vez establecidos los requerimientos iniciales, seleccionadas las primeras historias de usuario a implementar y haber modelado una idea general de cómo funcionaría el sistema, se planifican 5 iteraciones.

3.3 Iteración 1

- **Planificación**

Iteración 1	
Descripción	Presentación del espacio de la aplicación que el usuario utilizará para diseñar interfaces gráficas de usuario incluyendo los componentes disponibles y características editables.
Historia de Usuario a Desarrollar	1.- Elaborar interfaz de la aplicación web
	6.- Diseñar barra de componentes gráficos
	7.- Diseñar área de edición de propiedades
	8.- Agregar componentes
	9.- Eliminar componentes
Tiempo Estimado	16 días
Fecha Inicio/Fin	18-06-2007 / 03-07-2007

Tareas por Historia de Usuario

HU 1.- Elaborar interfaz de la aplicación web.

- Hacer diseño base de la página.
- Agregar contenido básico: título, logo, secciones, entre otros.

HU 6.- Diseñar barra de componentes gráficos.

- Elaborar ícono para cada componente gráfico.
- Ubicar barra lateral y organizar dentro de esta los componentes. Colocar tooltips de identificación a cada componente.

HU 7.- Diseñar área de edición de propiedades.

- Establecer las propiedades y eventos que se manejaran por cada componente.
- Definir un tipo de entrada en un formulario que permita editar cada propiedad del componente.
- Diseñar e implementar un proceso que permita construir un formulario con las propiedades a editar correspondiente a cada tipo de componente.

HU 8.- Agregar componentes.

- Establecer representación y correspondencia de los componentes gráficos en componentes web desarrollados con html.
- Crear componente directamente en el área de trabajo al ser seleccionado.
- Establecer y almacenar propiedades iniciales del componente creado.

HU 9.- Eliminar componentes.

- Borrar el componente del área de trabajo.
- Eliminar toda la información almacenada relacionada con el componente.

- **Diseño**

A partir de los componentes a implementar para el sistema, se investigó cuales de las propiedades comunes en las herramientas de construcción de GUIs Java son programables con Swing y los tipos de eventos (agrupados por oyente del evento) disponibles para cada uno. La ventana es un caso especial y todos los eventos que escucha se agrupan en el oyente *WindowListener.* A partir de los resultados se presentan en las tablas 8 y 9 los componentes, propiedades y eventos a implementar en el sistema:

	Nombre	**Valor**	**Tamaño**	**Posición**	**Alineación**	**Color**	**Tamaño Letra**	**Estilo Letra**	**Fuente Letra**	**Color Letra**	**Grupo**
Frame	x	x	X			x					
Button	x	x	X	x		x	x	x	x	x	
Label	x	x	X	x	x	x	x	x	x	x	
Text	x	x	x	x		x	x	x	x	x	
TextArea	x	x	x	x		x	x	x	x	x	
CheckBox	x	x	x	x		x	x	x	x	x	
RadioButton	x	x	x	x		x	x	x	x	x	x
ComboBox	x	x	x	x		x	x	x	x	x	

Tabla 8: Matriz de componentes y propiedades configurables

	Mouse	Foco	Teclas	Ventana
Frame				x
Button	x	x		
Label	x			
Text	x	x	x	
TextArea	x	x	x	
CheckBox	x	x		
RadioButton	x	x		
ComboBox	x	x	x	

Tabla 9: Matriz de componentes y eventos configurables

El resultado obtenido fue la base para la construcción de la primera versión del diagrama de clase del sistema compuesto por la clase *Componente* en la cual se agrupan todos los posibles atributos de un componente sin importar su tipo. Su representación se observa en la figura 16.

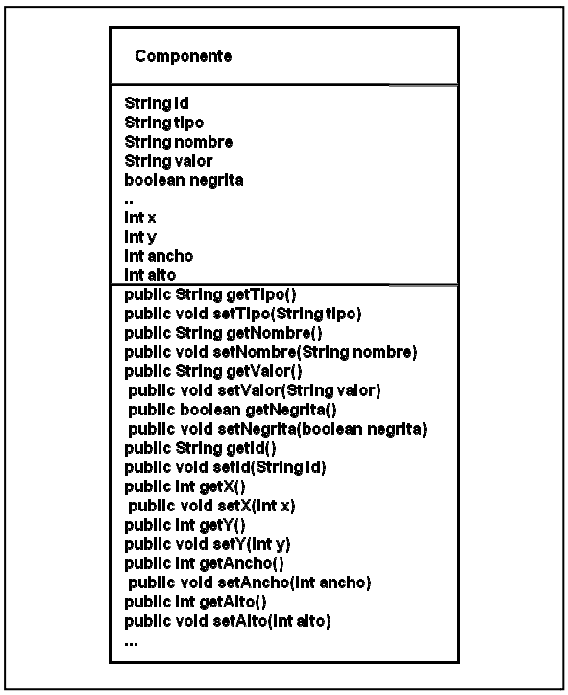


Figura16: Diagrama de Clases. Iteración 1

Además, para controlar la presentación y edición de las propiedades y eventos de un componente según su tipo, se creó una base de datos que los asocia con sus propiedades, y a éstas a su vez con las diferentes opciones y valores disponible (ver modelo de datos en la figura 17). Dicha información se completó con tipos de entrada en HTML para cada valor de una propiedad y eventos asociados para detectar cambios en ellos; todo con el fin de lograr la construcción de un formulario en el área de diseño que se ajuste al componente seleccionado según sus características.

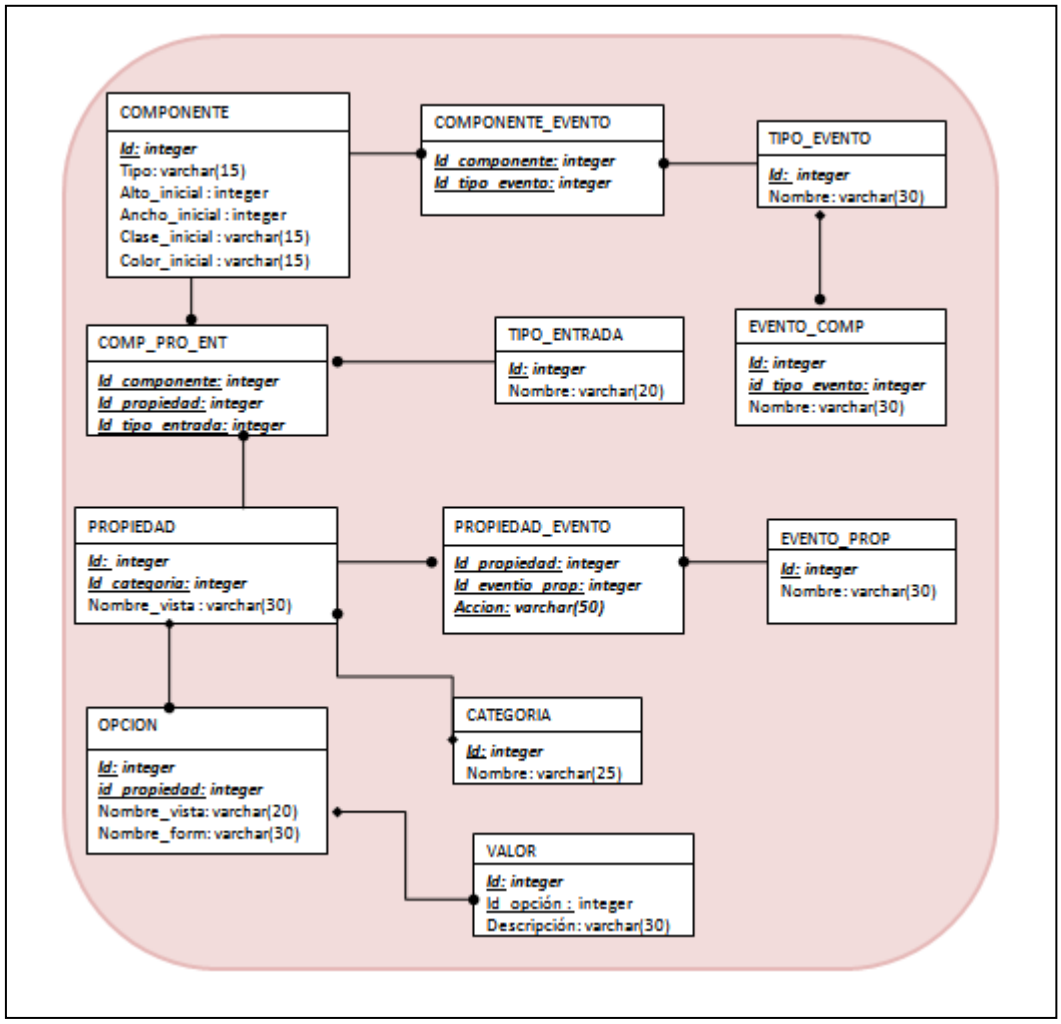


Figura 17: Modelo de datos. Iteración 1

Finalmente, para enviar los datos necesarios obtenidos de la base de datos para construir el formulario de propiedades y eventos del lado del cliente se estableció el formato XML que se muestra en las figura 18 (las propiedades de los componentes de dividieron en las categorías: general, apariencia y texto):

```
<formulario>
   <categorias>
        <categoria nombre=' '>
                <propiedad tipo=' '>

        <nombre_vista></nombre_vista>
                  <opciones>
                     <opcion nombre=' '
        evento=' ' accion=' ' etiqueta=' '>
                           <valor></valor>
                    </opcion>
                  </opciones>
                </propiedad>
        </categoria>
    </categorias>
    <eventos>
        <evento tipo=' '>
                <valor></valor>
        </evento>
     </eventos>
</formulario>
```

```
<!ELEMENT formulario (categorias, eventos?)>
<!ELEMENT categorias (categoria+)>
<!ELEMENT categoria (propiedad+)>
<!ATTLIST categoria nombre CDATA #REQUIRED>
<!ELEMENT propiedad (nombre_vista, opciones)>
<!ATTLIST propiedad tipo CDATA #REQUIRED>
<!ELEMENT nombre_vista (#PCDATA)>
<!ELEMENT opciones (opcion+)>
<!ELEMENT opcion (valor+)>
<!ATTLIST opcion nombre CDATA #REQUIRED>
<!ATTLIST opcion evento CDATA #REQUIRED>
<!ATTLIST opcion accion CDATA #REQUIRED>
<!ATTLIST opcion etiqueta CDATA #IMPLIED>
<!ELEMENT valor (#PCDATA)>
<!ELEMENT eventos (evento+)>
<!ELEMENT evento (valor+)>
<!ATTLIST evento tipo CDATA #REQUIRED>
<!ELEMENT valor (#PCDATA)>
```

Figura 18: Estructura XML de datos para la construcción del formulario de propiedades de un tipo de componente y su DTD correspondiente

- **Codificación**

La programación de las actividades que cubrían la iteración fue en HTML en principio para crear la interfaz web de la aplicación relacionada con el área de diseño.

Con respecto a la HU 8, ésta se dividió en 2 historias de usuario de menor alcance:

Número:8.1	*Nombre:* Construir componentes gráficos HTML	
Prioridad: Alta	*Riesgo en desarrollo:* Baja	*Estimación del tiempo de implementación:*2 días
Descripción: Se deben diseñar objetos manejables en HTML para que representen los componentes gráficos que conformarán la interfaz gráfica de usuario a construir		

Número: 8.2	*Nombre:* Añadir componente al área de trabajo	
Prioridad: Alta	*Riesgo en desarrollo:* Media	*Estimación del tiempo de implementación:*4 días
Descripción: Cuando el usuario seleccione un componente de la barra de herramientas, éste debe aparecer en el área de trabajo para poder comenzar a utilizarlo.		

Para el diseño de los componentes gráficos en “versión web” se utilizaron archivos CSS. En la figura 19 se pueden observar ejemplos del código CSS y su aplicación a un elemento HTML (div).

Código CSS	Resultado

.boton_html{
background-color: #cccccc;
font-family: verdana, arial;
font-size: 8pt;
height:25px;
width:90;
border: 2px;
border-style:groove;
border-color: #666666;
}
	

.combo_html{
background-color: #FFFFFF;
background:#fff url(../images/componentes_html/FLECHA_COMBO.PNG) no-repeat;
font-family: verdana, arial;
font-size: 8pt;
height:18px;
width:130px;
border: 2px ;
border-style:groove;
border-color:#FFFFFF;
}
	

Figura 19: Ejemplo de código CSS para crear componentes gráficos en HTML

El área de construcción de interfaces gráficas se compone de 5 secciones principales que se especifican en la figura 20: área de diseño, contenedor principal, paleta de componentes disponibles, menú de opciones relacionadas a los componentes y la interfaz y área de propiedades; las cuales fueron comunes en las herramientas generadoras de GUI analizadas en el Capítulo 2 y sus características fueron excelente guía de diseño:

- **Area de diseño:** Espacio disponible para realizar el diseño de la interfaz.
- **Contenedor principal:** Ventana contenedora del resto de todos los componentes que formarán parte del diseño.
- **Paleta de componentes:** Barra lateral que presenta los componentes disponibles para agregar al diseño.
- **Area de propiedades:** Barra lateral que presenta las propiedades editables de cada componente organizadas en categorías (su implementación se verá en la Iteración 2).
- **Menú de opciones:** Menú de diferentes acciones a realizar con los componentes y/o el diseño de GUI.

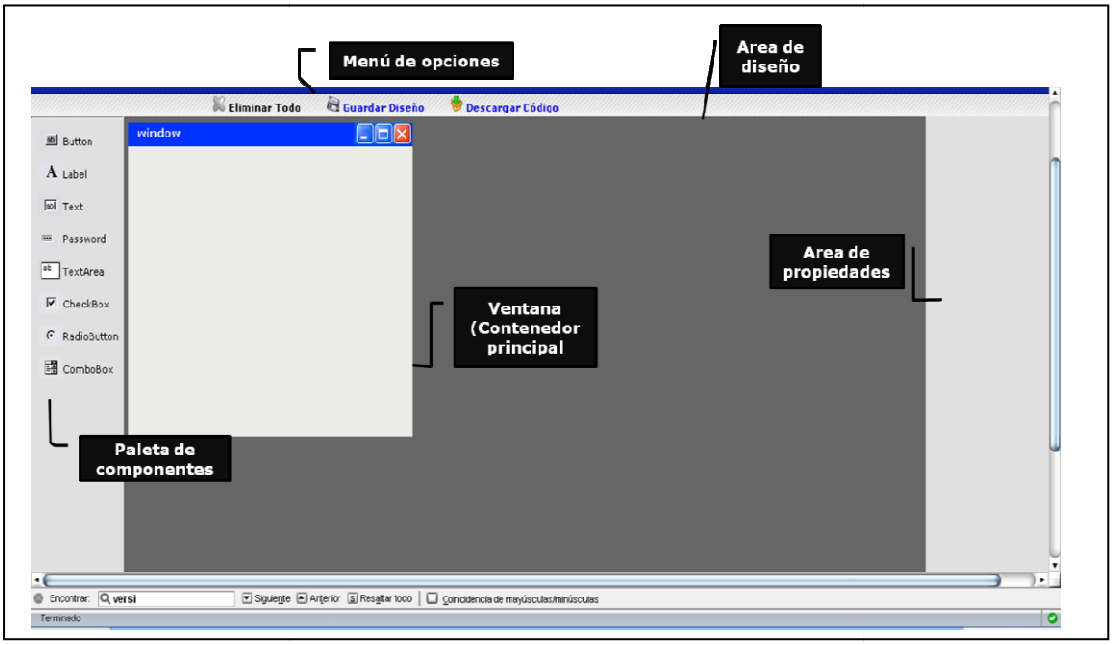


Figura 20: Ejemplo de vista general de la sección de diseño de la aplicación

En esta iteración, la historia de usuario de mayor riesgo de implementación fue la número 7. El proceso resultado para construir un formulario adecuado a cada tipo de componente fue solicitarlo desde el cliente y recibir data con el formato XML presentado en la figura 18 con toda la información necesaria para su construcción, validación y edición que se mantiene en sesión para su rápido acceso al ser solicitado para el mismo u otros componentes del mismo tipo.

```
function construirFormulario(){
    ..
        try{
        //Procesar xml para construir formulario
        var xmldoc = xml_http.responseXML;
        var root_node = xmldoc.getElementsByTagName('formulario');
        //Agregar las propiedades por categoria
        for( var i=0; i < xmldoc.getElementsByTagName('categoria').length; i++){
                html_form=html_form+"<tr>"
                html_form=html_form+"<td width='86' colspan='2' style='vertical-align:top;'>"
                html_form=html_form+"<hr></hr><span><b>"+xmldoc.getElementsByTagName('categoria').item(i).attrib
        utes.getNamedItem("nombre").value;+"</b></span>"
                html_form=html_form+"</td>"
                html_form=html_form+"</tr>"
                var propiedades = xmldoc.getElementsByTagName('categoria')[i].childNodes;
                for (var j = 0; j < propiedades.length; j++){
                  ..
            switch (tipo_entrada){
                  case "text": html_form=html_form+"<td width='120'><input name='"+nombre_prop_form+"'
```

```
            id='"+nombre_prop_form+"' type='"+tipo_entrada+"' class='texto_propiedades' size='20'
            "+evento_act+"='"+accion_act+"'></td>";
                        break;
                        case "combo":html_form=html_form+"<td><select name='"+nombre_prop_form+"'
            id='"+nombre_prop_form+"' style='font-size:9px' "+evento_act+"='"+accion_act+"'>";
                        html_form=html_form+"<option value='' selected='selected'></option>"
                        var opciones = propiedades[j].getElementsByTagName('valor');
                        for (var k = 0; k < opciones.length; k++)
                        {..
}..
```

Figura 21: Parte del código Javascript encargado de construir formulario

- **Pruebas**

Las pruebas unitarias y de aceptación relacionadas a las primeras historias de usuario de esta iteración fueron pocas y sencillas, verificando unicidad en la apariencia de la interfaz de la aplicación en general, colores y características de los componentes gráficos HTML. Con respecto a la creación de área de edición de propiedades se hicieron pruebas del lado del cliente y del servidor para crear los posibles diferentes formularios para los tipos de componente.

Pruebas más importantes del lado del cliente:

- Tooltips de identificación en barra de componentes
- Verificar formulario de propiedades al seleccionar un botón (se repite para cada componente).

Id	Modulo	Descripción de la Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
1	Interfaz Web	Tooltips de identificación en barra de componentes	Tipo de componente señalado con el mouse	Tipo de componente señalado con el mouse	
2	Interfaz Web	Verificar formulario de propiedades al seleccionar un botón	Formulario con las propiedades de la ventana dibujado en la barra lateral derecha del área de diseño	Formulario con las propiedades de la ventana dibujado en la barra lateral derecha del área de diseño	
6	Interfaz Web	Verificar formulario de propiedades al seleccionar una caja de selección	Formulario con las propiedades de la caja de selección dibujado en la barra lateral derecha del área de diseño	Faltó campo para seleccionar si la caja aparecerá seleccionada o no	Falto incluir datos de la propiedad en la base de datos

Tabla 10: Algunas pruebas unitarias del lado del cliente en la Iteración 1

Pruebas más importantes del lado del servidor (métodos por clase):

Clase: ManejadorComponente

Método CrearXMLFormulario:

- Verificar que no retorne una cadena vacía.
- Verificar validez del XML de retorno.

El método *crearXMLFormulario* recibe como parámetro el tipo de componente seleccionado, para el cual se debe construir data XML que permite construir el formulario de propiedades correspondientes del lado del cliente; las pruebas implementadas para este método toman como entrada un tipo de componente específico y verifican que el método no genere como salida una cadena vacía (ver código asociado en la figura 22) y que contenga las propiedades indicadas para dicho componente.

```
public void testCrearXMLFormulario(){
        try{

                String xml = mc.crearXMLFormulario("button");
                //Probar que no  devuelva un string vacío
                //assertEquals("Devuelve un string vacio", xml.length(), 0);

                int tam = xml.length();
                if (tam <= 0) fail("Devolvio algo vacio");
        }catch(Exception e){
                assertTrue("No abrio la bd", true);
        }
}
```

Figura 22: Ejemplo de prueba unitaria con JUnit en la Iteración 1

En la siguiente figura se pueden observar diferentes contenidos del área de propiedades presentada al usuario al seleccionar algunos de los componentes gráficos

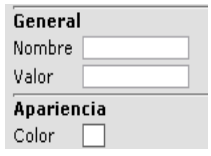
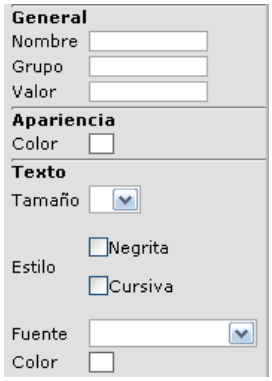
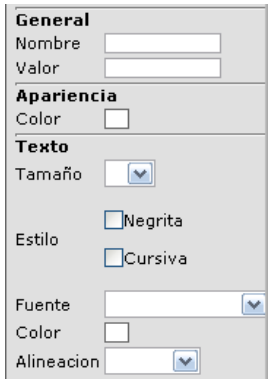
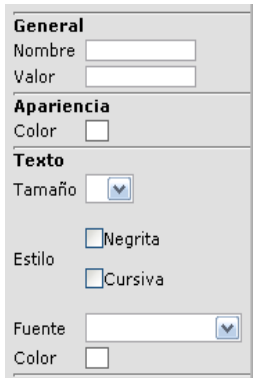
Ventana	Botón de opción	Etiqueta	Botón
			

Figura 23: Barras de propiedades para algunos de los componentes

- **Observaciones**

La HU 8 no pudo completarse, de manera tal que las tareas faltantes pasaron a formar parte de la siguiente iteración al igual que la HU 9. En este momento se revisó la planificación inicial para dividir aquellas historias de usuario que pudiesen pasar por la misma situación y así mejorar la organización y agenda de desarrollo. Así, se dividieron las historias 10 y 11 de la siguiente manera:

HU 10.- Editar componentes

Número: 10.1	*Nombre:* Editar propiedades visuales.	
Prioridad: Alta	*Riesgo en desarrollo:* Alto	*Estimación del tiempo de implementación:* 3 días
Descripción: El usuario podrá editar las características gráficas visuales de cada componente como su color, valor y el color, estilo y tamaño de la fuente de su contenido.		

Número: 10.2	*Nombre:* Editar propiedades de espacio.	
Prioridad: Alta	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 4 días
Descripción: El usuario podrá editar tamaño y ubicación de cada componente.		

Número: 10.3	*Nombre:* Asignar eventos	
Prioridad: Media	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 3 días
Descripción: El usuario podrá elegir a cuales eventos va a responder un componente y para que se incluya su estructura en el código fuente.		

HU 11.- Generar código fuente

Número: 11.1	*Nombre:* Recolectar información.	
Prioridad: Alta	*Riesgo en desarrollo :*Alta	*Estimación del tiempo de implementación:* 3 días
Descripción: El sistema debe obtener todos los datos de la interfaz diseñada para generar su código cuando el usuario indique que así lo desea.		

Número:11.2	*Nombre:* Transformación de data	
Prioridad: Alta	*Riesgo en desarrollo:* Alta	*Estimación del tiempo de implementación:* 6 días
Descripción: El sistema debe procesar eficientemente los datos para crear el código fuente de la interfaz diseñada por el cliente en la página en lenguaje Java.		

3.4 Iteración 2

- **Planificación**

Iteración 2	
Descripción	Implementar la manipulación sencilla de los componentes, permitiendo la ejecución de 3 acciones principales: agregar, eliminar y editar componentes gráficos al diseño; incluyendo en la edición el manejo de las propiedades visuales y espaciales (tamaño y ubicación) de los componentes.
Historia de Usuario a Desarrollar	8.2.- Añadir componente al área de trabajo
	9.- Eliminar componentes
	10.2.- Editar propiedades de espacio
	10.1.- Editar propiedades visuales
Tiempo Estimado	13 días
Fecha Inicio/Fin	04-07-2007 / 20-07-2007

Tareas por Historia de Usuario

HU 8.2.- Añadir componente al área de trabajo

- Crear componente directamente en el área de trabajo al ser seleccionado.
- Establecer y almacenar propiedades iniciales del componente creado.

HU 9.- Eliminar componentes.

- Eliminar toda la información almacenada relacionada con el componente.
- Borrar el componente del área de trabajo.

HU 10.2.- Editar propiedades de espacio.

- Hacer de cada componente un objeto manipulable con eventos del mouse.
- Establecer restricciones de cambios de tamaño.
- Establecer restricciones de movimiento.

HU 10.1.- Editar propiedades visuales.

- Programar envío de información al servidor para su actualización.
- Implementar recepción de confirmación y actualizar valor de la propiedad actualizada en el formulario de propiedades y en las características del componente si aplica.

- **Diseño**

Tomando en cuenta que el almacenamiento de las propiedades de los componentes integrantes del diseño en creación estaría localizado del lado del servidor, se crearon diferentes formatos necesarios para procesar la información necesaria para la creación, edición y eliminación de los mismos del servidor hacia el cliente, siendo éste último responsable de reflejar los resultados de dichas acciones directamente en el diseño. Los diferentes modelos de data XML se muestran a continuación en las figuras 24 a la 26.

```
<componente id=' ' tipo=' '>
        <ancho_inicial></ancho_inicial>
        <alto_inicial></alto_inicial>
        <clase_inicial></clase_inicial>
</componente>
```

```
<!ELEMENT componente (ancho_inicial, alto_inicial, clase_inicial)>
<!ATTLIST componente id CDATA #REQUIRED>
<!ATTLIST componente tipo CDATA #REQUIRED>
<!ELEMENT ancho_inicial (#PCDATA)>
<!ELEMENT alto_inicial (#PCDATA)>
<!ELEMENT clase_inicial (#PCDATA)>
```

Figura 24: Estructura XML de datos para dibujar inicialmente un componente en el área de diseño y su DTD correspondiente

```
<componente id=' ' tipo=' '>
  <propiedades>
    <propiedad tipo=' ' nombre=' ' valor=' '/>
  </propiedades>
  <eventos>
    <evento tipo=' ' valor=' '/>
  </eventos>
</componente>
```

```
<!ELEMENT componente (propiedades, eventos?)>
<!ATTLIST componente id CDATA #REQUIRED>
<!ATTLIST componente tipo CDATA #REQUIRED>
<!ELEMENT propiedades (propiedad+)>
<!ELEMENT propiedad EMPTY>
<!ATTLIST propiedad tipo CDATA #REQUIRED>
<!ATTLIST propiedad nombre CDATA #REQUIRED>
<!ATTLIST propiedad valor CDATA #REQUIRED>
<!ELEMENT eventos (evento+)>
<!ELEMENT evento EMPTY>
<!ATTLIST evento tipo CDATA #REQUIRED>
<!ATTLIST evento valor (si |no) "no">
```

Figura 25: Estructura XML de datos para recibir valor de las propiedades y eventos de un componente seleccionado o actualizado

```
<mensaje>
   <respuesta></respuesta>
</mensaje>
```

```
<!ELEMENT mensaje (respuesta)>
<!ELEMENT respuesta (#PCDATA)>
```

Figura 26: Estructura XML de datos para mensaje de error por fallas en el servidor y su DTD correspondiente

- **Codificación**

Para agregar un componente, se inicializan las características del mismo con los datos provenientes de la base de datos del sistema dependiendo de su tipo (ver figura 27). Luego, el manejo de su ubicación y tamaño se realizó sólo con librerías Javascript, haciendo uso constante del DOM de la página donde se lleva a cabo el diseño.

```
public Componente crearComponente(String id, String tipo, String nombre) throws Exception
        {
                Componente c= new Componente();
                ManejadorBD baseDatos = new ManejadorBD();
                String sql = "";
                ResultSet rset;

                c.setId(id);
                c.setTipo(tipo);
                c.setColor_letra("000000");
                c.setCursiva(false);
                c.setFuente("Verdana");
                ..
                c.setValor(tipo);
                c.setSeleccionado(false);
                c.setMouse(false);
                c.setFoco(false);
                c.setTexto(false);
                c.setTeclas(false);
                c.setVentana(false);

                //Obtener el tamaño incial y el color inicial del componente
                //Consultar el componente en la BD
                sql = "SELECT alto_inicial, ancho_inicial, color_inicial, clase_inicial FROM componente where
tipo = '" + tipo + "'";
                rset = baseDatos.ejecutarConsulta(sql);

                if(rset.first()){
                        //Colocar los valores iniciales obtenidos
                        c.setAlto(rset.getInt("alto_inicial"));
                        ..
                        c.setColor_fondo(rset.getString("color_inicial"));
                        c.setClase(rset.getString("clase_inicial"));
                }
                return c;
        }
```

Figura 27: Parte del código encargado de inicializar el componente a agregar en el diseño

Fue un proceso complicado lograr controlar la selección de un componente específico a la vez gracias a la superposición de algunos sobre otros en la pantalla, por ejemplo los componentes sobre la ventana, ya que cualquier selección (click) sobre un componente automáticamente incluía a los elementos inferiores a éste, repitiendo el evento. Era necesario identificarlos por tipo de componente para controlar los tamaños permitidos y en general para que no sobrepasaran los límites de la ventana que los contiene.

Para el cambio de tamaño se incluye a cada componente 8 delimitadores que se posicionan en las equinas y puntos medios entre laterales del componente y pasan a formar parte de éste. A partir de ellos se consultan y controlan los eventos y las variables esenciales para controlar los cambios de tamaño y ubicación del componente al cual pertenecen, estas variables son: posición(x,y) y tamaño(ancho,alto). En la figura 28 se observa un componente con sus delimitadores; cabe destacar que las características visuales y comportamiento del cursor sobre ellos se programó con CSS.

Código fuente Javascript	Ejemplo resultado

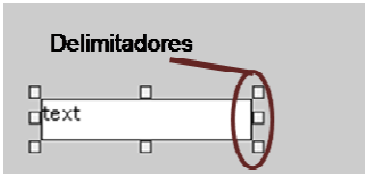
DragResize.prototype.resizeHandleSet = function(elm, show) { with (this)
{
 	
if (!elm._handle_tr)
{
for (var h = 0; h < handles.length; h++)
{
var hDiv = document.createElement('div');
hDiv.className = myName + ' ' + myName + '-' + handles[h];
hDiv.id="cuadrito";
hDiv.onclick=function
(e){obtenerIdComponenteActual(e.target.getAttribute("id"),"cuadrito");};
elm['_handle_' + handles[h]] = elm.appendChild(hDiv);
}
}
 	
for (var h = 0; h < handles.length; h++)
{
elm['_handle_' + handles[h]].style.visibility = show ? 'inherit' : 'hidden';
}
}};
	

Figura 28: Código genérico para inclusión de delimitadores

Para la edición de propiedades visuales de los componentes gráficos en la aplicación se utilizó como recurso de almacenamiento temporal la sesión del usuario en el sistema; y es allí donde se almacenan temporalmente los componentes como objetos con todos sus datos y se manipulan fácilmente. Cada componente se guarda en la sesión identificado unívocamente y se acceden a ellos cuando el cliente realiza una petición generada por la selección o actualización de éste por el usuario, enviando los parámetros según la acción y recibiendo data XML de respuesta (Modelo de XML figura 25).

```
function actualizarComponente(){
        //Ubicar formulario de propiedades
        var propiedades = document.form_propiedades;
        ..
        //Recorrer campos del formulario para tomar las propiedades del componente seleccionado
        for (var i=1; i<propiedades.length; i++)
        {
           //Contruir url con todas las caracteristica del componente
            if( propiedades.elements[i].type=="checkbox"){
                  url=url+ "&"+propiedades.elements[i].name+"="+propiedades.elements[i].checked;
             }else{
                  url = url + "&"+propiedades.elements[i].name+"="+propiedades.elements[i].value;
             }
        }
         ..
   //Envío de datos al servidor con AJAX
        xml_http.open("GET", url, true);
        xml_http.send(null);
}
```

Figura 29: Parte del código Javascript para realizar envío de propiedades del componente

- **Pruebas**

Las pruebas de aceptación con el cliente se realizaron poco después de finalizada la iteración y en ellas surgieron errores relacionados a los valores de las propiedades de los componentes y su repercusión en los mismos visualmente.

Pruebas más importantes del lado del cliente:

- Muestra de propiedades iniciales en la barra.
- Cambio de las propiedades al alternar entre componentes y la ventana.
- Eliminación de un componente de la interfaz.
- Combinar acciones: eliminar un elemento y eliminarlos todos.
- Movimiento de un componente en el área permitida.
- Cambio de tamaño limitado según su tipo de componente.
- Cambio de tamaño dentro del límite de la ventana que lo contiene.
- Cambio de la propiedad *fuente de la letra* (se repite la prueba para cada propiedad).
- Cambio de la propiedad *nombre* de la ventana (se repite la prueba para cada propiedad).
- Actualización de ubicación y tamaño al terminar el movimiento del mouse con el componente.

Id	Modulo	Descripción de Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
13	Manejo de componentes gráficos	Cambio de las propiedades al alternar entre componentes y la ventana	Al alternar entre componentes y la ventana se cambien las propiedades en el formulario de acuerdo al caso	Cambian las propiedades acordes a la ventana pero desaparecen los componentes.	La ventana se sobrepone a los componentes al ser seleccionada
15	Manejo de componentes gráficos	Eliminación de un componente de la interfaz	Desaparece el elemento de la pantalla y los valores de sus propiedades	Desaparece el elemento de la pantalla pero no se borran las propiedades de la pantalla	Faltó implementar función para limpiar los valores de las propiedades de la pantalla.
17	Manejo de componentes gráficos	Movimiento de un componente en el área permitida	El componente debe moverse sólo dentro del área que ocupa la ventana que lo contiene	El componente se mueve por todo el espacio de la pantalla	El componente toma la identificación de la ventana que lo contiene y se desconfigura su comportamiento
18	Manejo de componentes gráficos	Cambio de tamaño cualquiera de un componente	Cambio del tamaño generado por el movimiento del mouse	Componente trasforma su tamaño tomando las propiedades de la ventana	Superposición de identificadores de la ventana por ser capa contenedora de los mismos

Tabla 11: Algunas pruebas unitarias del lado del cliente en la Iteración 2

Pruebas más importantes del lado del servidor (métodos por clase):

Clase: ManejadorComponente

Método CrearComponente:

- Verificar que no retorne como respuesta objetos vacíos.
- Verificar que al crear el componente no contenga datos erróneos.

Método ObtenerPropiedades

- Verificar que dado un componente no retorne como respuesta un arreglo de propiedades vacío.

Método AsignarPropiedad

- Verificar que se modifique el valor del componente deseado.

Método CrearXMLComponente

- Verificar que no retorne como respuesta una cadena vacía en lugar del XML.
- Verificar que el XML contenga la información propia del componente.

El método *crearComponente* recibe como parámetros, el identificador del componente, tipo y nombre; con esto construye un objeto de la clase *Componente* que contiene las propiedades que maneja dicho componente. Las pruebas implementadas para esté método verifica que la salida no sea un objeto nulo al

pasarle un tipo de componente válido y que maneje las propiedades correctas: (ver código asociado en la figura 30).

```
public void testCrearComponente(){
        Componente c1 = null;
        Componente c2 = null;
        try{
                c2 = mc.CrearComponente("c1", "button", "boton1");
        }catch(Exception e){
                e.printStackTrace();
                assertTrue("No abrio la bd", false);
        }

        System.out.println("Color Incial: " + c2.getColor_fondo());

        //pruebo que no me devuelva un objeto nulo
        assertNotNull("funciona bien",c2);
        //assertNull("funciona bien", c2);

        //pruebo que no me devuelva datos vacíos o erróneos
        assertEquals("No devuelve lo que tiene que devolver", c2.getAlineacion(), "center");
        assertEquals("No devuelve lo que tiene que devolver", c2.getOnclick(), false);

        }
```

Figura 30: Ejemplo de prueba unitaria con JUnit en la Iteración 2

- **Observaciones**

Durante las pruebas de aceptación con el cliente, éste realizó correcciones acerca de la apariencia del área del diseño dando recomendaciones para mejorarla y ofrecer más espacio al usuario; además surgió la idea una funcionalidad del sistema en el área del sistema que permita al usuario eliminar todo el trabajo realizado hasta el momento, para lo cual se añadió una nueva HU que se incluyó para la siguiente iteración por su relación a las actividades de edición de diseño:

Número: 14	*Nombre:* Limpiar área de trabajo.	
Prioridad: Media	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 2 días
Descripción: El usuario podrá eliminar todos los componentes del área de trabajo y comenzar de nuevo su diseño.		

3.5 Iteración 3

- **Planificación**

***Iteración* 3**	
Descripción	Implementación del resto de las funcionalidades requeridas: asignar eventos a cada componente y la opción de borrar todo el diseño de la ventana. Además, implementar el primer paso de generación del código fuente que consiste en recolectar toda la información de la interfaz y organizarla en formato XML.
Historia de Usuario a Desarrollar	10.3.- Asignar eventos
	14.- Limpiar área de trabajo
	11.1.- Recolectar información de la interfaz
Tiempo Estimado	8 días
Fecha Inicio/Fin	23-07-2007 / 03-08-2007

Tareas por Historia de Usuario

HU 10.3.- Asignar eventos

- Diseño de sección única para selección de eventos dentro de las propiedades.
- Implementar procedimientos para mostrar, agregar y eliminar eventos para un componente.
- Actualizar eventos en el servidor.

HU 14.- Limpiar área de trabajo

- Diseñar y ubicar icono que identifique la nueva funcionalidad.
- Eliminar componentes del lado del cliente y del lado del servidor.

HU 11.1.- Recolectar información de la interfaz.

- Diseñar formato XML para estructurar datos de la interfaz.
- Recorrer almacenamiento temporal de información de componentes y crear XML.
- Guardar resultado en un archivo físico en el servidor.

- **Diseño**

Swing maneja los eventos de un componente agrupados por su oyente o *ActionListener* (los contiene como interfaces), razón por la cual en el código fuente se deben definir todos los eventos por oyente aunque se necesite uno solo. Por lo tanto se le ofrecerá al usuario la implementación de eventos según su oyente:

Tipo de Evento (por oyente)	Evento
Mouse (ratón)	OnClick OnMouseOver OnMouseOut OnMousePressed OnMouseReleased
Focus (Foco)	OnFocusGained OnFocusLost
Key (Teclado)	OnKeyPressed OnKeyReleased
Window (Ventana)	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened .

Tabla 12: Clasificación de eventos más utilizados por oyente

Para la realización de la HU 11.1 se construyó el formato XML de la figura 31 en el que se recoge toda la información de los componentes que forman la interfaz de usuario elaborada por el usuario y se diseño con la intención que fuera reutilizable para el momento de hacer alguna otra acción sobre la GUI completa, como por ejemplo recuperarla después de haberla diseñado:

```
<gui nombre_clase=''>
   <componentes>
      <componente id='' tipo="">
        <propiedades>
           <propiedad nombre=" valor="/>
        </propiedades>
        <eventos>
           <evento tipo=" valor=''/>
        </eventos>
      </componente>
   </componentes>
  <complementos>
      <complemento tipo=" nombre=" >
        <propiedades>
           <propiedad nombre=" valor="/>
        </propiedades>
      </complemento>
</complementos>
</gui>
```

```
<!ELEMENT gui (componentes, complementos?)>
<!ATTLIST gui nombre_clase CDATA #REQUIRED>
<!ELEMENT componentes (componente+)>
<!ELEMENT componente (propiedades,eventos?)>
<!ATTLIST componente id CDATA #REQUIRED>
<!ATTLIST componente tipo CDATA #REQUIRED>
<!ELEMENT propiedades (propiedad+)>
<!ELEMENT propiedad EMPTY>
<!ATTLIST propiedad nombre CDATA #REQUIRED>
<!ATTLIST propiedad valor CDATA #REQUIRED>
<!ELEMENT eventos (evento+)>
<!ELEMENT evento EMPTY>
<!ATTLIST evento tipo CDATA #REQUIRED>
<!ATTLIST evento valor CDATA #REQUIRED>
<!ELEMENT complementos (complemento+)>
<!ELEMENT complemento (propiedades)>
<!ATTLIST componente tipo CDATA #REQUIRED>
<!ATTLIST componente nombre CDATA #REQUIRED>
<!ELEMENT propiedades (propiedad+)>
<!ELEMENT propiedad EMPTY>
<!ATTLIST propiedad nombre CDATA #REQUIRED>
<!ATTLIST propiedad valor CDATA #REQUIRED>
```

Figura 31: Estructura XML de datos para recolectar datos de todos los componentes que conforman la interfaz y su DTD correspondiente

- **Codificación**

Los eventos se manejaron como el resto de las propiedades del componente: se agregaron al área de propiedades en una pequeña ventana que contendrá los eventos disponibles dependiendo del componente seleccionado y al ser seleccionado o eliminado para éste, la información para su actualización viajará como parte de la data XML que contiene las características del componente actualizado al servidor (ver figura 25), su presentación se observa en la figura 32.

Con respecto a la eliminación de todos los componentes de la ventana, este requerimiento se pudo implementar en menos tiempo que el establecido (código fuente asociado se muestra en la figura 33).

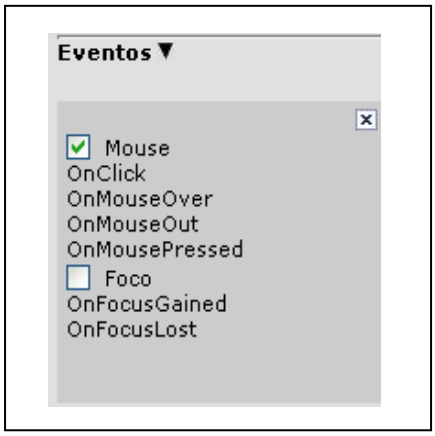


Figura 32: Inclusión de eventos en las propiedades del componente

```
public class EliminarComponente extends HttpServlet{

        public void service(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {

                //Tomar del url el tipo de componente a obtener sus propiedades
                String idComponente=request.getParameter("id");

                //Buscar componente en la sesion
                HttpSession session= request.getSession();
                session.removeAttribute(idComponente);

                //Devolver respuesta al navegador
                response.setContentType("text/xml");
                response.setHeader("Cache-Control", "no-cache");
                response.getWriter().write(idComponente);

   }
}
```

Figura 33: Parte del servlet encargado de eliminación de un componente

Para finalizar las historias de usuario de esta iteración, en la recolección de datos de la interfaz se implementó una función que recorre todos los componentes del diseño y se encarga de tomar los valores de posición y tamaño para llevarlos al servidor junto con los demás y poder completar la información necesaria para la generación del código fuente asociado a dicho diseño, crear el XML asociado y almacenarlo con el nombre que el usuario elija para su diseño.

- **Pruebas**

Las pruebas unitarias del lado del servidor se realizaron sobre los métodos encargados de la creación de la data XML con la información de la GUI y el manejo de archivos. Del lado del cliente se realizaron una variedad de pruebas con asignación y eliminación de eventos para verificar llamadas al servidor y correctas actualizaciones.

Pruebas más importantes del lado del cliente:

- Asignación de eventos a los componentes.
- Eliminación de eventos a los componentes.
- Limpiar área de trabajo completa.
- Recolección de ubicación y tamaño de todos los componentes.

Id	Modulo	Descripción de la Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
37	Manejo de componentes gráficos	Limpiar área de trabajo completa	Desaparecen todos los componentes de la ventana y sus registros en el servidor	No se recibe respuesta del servidor	Problema con el recorrido de la estructura de almacenamiento en el servidor
38	Modulo de Generación de Código Fuente	Recolección de ubicación y tamaño de todos los componentes	Actualización en el servidor de cada componente enviando parámetros correctos	Actualización en el servidor de cada componente enviando parámetros correctos	

Tabla 13: Algunas pruebas unitarias del lado del cliente en la Iteración 3

Pruebas más importantes del lado del servidor (métodos por clase):

Clase*: ManejadorComponente*

Método *ObtenerEventos*

- Verificar que dado un componente no retorne como respuesta un arreglo vacío en lugar de los eventos asociados al componente.

Método *AsignarEvento*

- Verificar que se modifique el valor del componente deseado.

Método *CrearXMLFinal*

- Verificar que no retorne como respuesta una cadena vacía en lugar del XML de la GUI.
- Verificar que contenga la información de todos los componentes.

El método *obtenerEventos* recibe como parámetro el tipo de componente seleccionado y devuelve un arreglo con todos los eventos que éste soporta, para este método se verifica que la salida no sea un arreglo vacío y que efectivamente los eventos del arreglo correspondan al componente indicado (ver código fuente asociado en la figura 34).

```
public void testObtenerEventos(){
        try{

                String[] eventos = mc.obtenerEventos("button");

                System.out.println("Cantidad de eventos:" + eventos.length);

                //Verificar que no  devuelva un string vacío
                assertEquals("Devuelve un string vacio", eventos.length, 8);

                }catch(Exception e){
                assertTrue("Problemas con el ManejadorComponentes", true);
        }
}
```

Figura 34: Ejemplo de prueba unitaria con JUnit en la Iteración 3

- **Observaciones**

Cabe destacar que durante la reunión para las pruebas de aceptación el cliente dio sugerencias para mejorar el cuadro de edición de eventos; y discutiendo sobre los módulos del sistema por desarrollar surge la propuesta de incluir una sección en la aplicación donde el usuario pueda actualizar sus datos personales.

Número: 15	*Nombre:* Actualizar datos personales.	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 día
Descripción: El cliente debe poder cambiar los datos que ingresó al registrarse en la página.		

3.6 Iteración 4

- **Planificación**

***Iteración* 4**	
Objetivo	Aplicar una transformación XSL al archivo en formato XML resultante de estructurar todos los datos de la interfaz diseñada por el usuario y generar código fuente Java coherente asociado al contenido.
Historia de Usuario a Desarrollar	11.2 Transformación de data
	12 Obtención de archivo java
Tiempo Estimado	9 días
Fecha Inicio/Fin	06-08-2007 / 16-08-2007

Tareas por Historia de Usuario

HU 11.2.- Transformación de data

- Establecer las técnicas de codificación a utilizar en el archivo Java a generar (declaración de variables, métodos, comentarios entre otros).
- Crear archivo XSL para realizar la transformación al archivo XML ya creado.
- Generar un archivo .java para la interfaz de usuario diseñada.

HU 12.- Obtención de archivo

- Presentar ventana emergente al usuario que le permita obtener el código java generado.

- **Codificación**

El archivo XSL para la transformación y generación del código fuente, contenido del archivo java a generar, se creó después de revisar varias guía de programación Swing para hacer las instrucciones de manera legible, sencilla y siguiendo los estándares Java presentados en el Capítulo II de éste documento. Sin embargo el tiempo de duración de la iteración se extendió en varios días por problemas de implementación de la transformación.

```
..
<xsl:template match="gui">
import javax.swing.*;
..
public class <xsl:value-of select="@nombre_clase"/> extends JFrame{
<xsl:for-each select="componentes">
<xsl:for-each select="componente">
<xsl:if test="@tipo = 'window'">
//Declaracion del marco principal de la ventana
private JFrame <xsl:for-each select="propiedades">
<xsl:for-each select="propiedad">
<xsl:if test="@nombre = 'nombre'"><xsl:value-of select="@valor"/> = new JFrame("</xsl:if>
</xsl:for-each>
</xsl:for-each>
<xsl:for-each select="propiedades">
<xsl:for-each select="propiedad">
<xsl:if test="@nombre = 'valor'"><xsl:value-of select="@valor"/>");
//Declaracion de variables represetantes de los componentes de la ventana
..
<xsl:for-each select="componentes">
<xsl:for-each select="componente">
<xsl:if test="@tipo != 'window'">
<xsl:choose><xsl:when test="@tipo = 'button'">
private JButton </xsl:when>
<xsl:when test="@tipo = 'label'">
..
<xsl:for-each select="propiedades">
<xsl:for-each select="propiedad">
<xsl:if test="@nombre = 'nombre'">String valores_<xsl:value-of select="@valor"/>[] = </xsl:if>
</xsl:for-each>
</xsl:for-each>
<xsl:for-each select="propiedades">
<xsl:for-each select="propiedad">
<xsl:if test="@nombre = 'valor'">{<xsl:value-of select="@valor"/>};</xsl:if>
</xsl:for-each></xsl:for-each>
..
private JPanel contentPane;
public void JVentana() {
..
//Propiedades de texto, tipo de letra, estilo de la letra y tamano
..
<xsl:if test="propiedades/propiedad[@nombre = 'negrita' and @valor='true']">
<xsl:if test="propiedades/propiedad[@nombre = 'cursiva' and @valor='false']">Font.BOLD,</xsl:if></xsl:if>
<xsl:if test="propiedades/propiedad[@nombre = 'negrita' and @valor='true']"><xsl:if
test="propiedades/propiedad[@nombre = 'cursiva' and
@valor='true']">Font.BOLD+Font.ITALIC,</xsl:if></xsl:if>
</xsl:for-each>
</xsl:for-each>
..
//Implementacion de eventos
..
<xsl:if test="eventos/evento[@tipo = 'Mouse' and @valor='true']">
<xsl:for-each select="propiedades">
<xsl:for-each select="propiedad">
<xsl:if test="@nombre = 'nombre'">
<xsl:value-of select="@valor"/>.addMouseListener(new MouseListener(){
        public void mouseEntered(MouseEvent e){        }
        ..
```

Figura 35: Extracto del archivo XSL a aplicar para generar código fuente Java de una GUI diseñada en el sistema

Finalmente, la transformación se hace del lado del servidor y el resultado se envía como respuesta especificando que su formato será un archivo (para que el browser inicie la actividad de descarga), dicho archivo lleva un nombre establecido al hacer la petición de generación de código fuente y es también el nombre de la clase java.

```
public class GenerarCodigo extends HttpServlet {

        public void service(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
        ..
        try{
                direccionXml=ma.crearDireccionXML(archivoXML);
                direccionXsl =ma.obtenerDireccionXSL();

                /*Antes de hacer la transformacion, configurar datos de la respuesta*/
                response.setContentType("text/xml");
                response.setHeader("Cache-Control", "no-cache");
                String archivoJava=archivoXML.replace(".xml", ".java");
                response.setHeader("Content-Disposition", "attachment; filename=\"" + archivoJava + "\"");

                /*Hacer transformación*/

                File estilo_local =new File(direccionXsl);
                File xml_local =new File(direccionXml);

                TransformerFactory tFactory = TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer(new StreamSource(estilo_local));
                transformer.transform(new StreamSource(xml_local), new StreamResult(response.getWriter()));
        }
                catch (Exception e) {
                  e.printStackTrace();
                }
                ..
        }
}
```

Figura 36: Parte del servlet encargado de realizar la transformación XSL y generar archivo .java

- **Prueba**

En esta iteración se realizaron pruebas sobre el comportamiento del sistema al momento de hacer la petición de generar el código de la interfaz diseñada y la forma de mostrar los resultados generados.

Además se mantuvo durante todas las pruebas un archivo de código fuente Java asociado a una interfaz gráfica implementada en Swing formada por un ejemplar de cada componente, y así poder comparar el código generado por la aplicación y verificar su correcta estructura, orden y contenido.

Pruebas más importantes del lado del cliente

- Mostrar al usuario mensajes de confirmación al pedir generar código.
- Mostrar al usuario cuadro con entrada de texto para solicitar nombre de archivo si no lo ha guardado antes.
- No mostrar al usuario cuadro con entrada de texto para solicitar nombre de archivo si ha guardado antes.
- Abrir ventana emergente del browser con el contenido del código generado.

Id	Modulo	Descripción de la Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
40	Proceso de transformación XSL	Mostrar al usuario mensajes de confirmación al pedir generar código	Ventana para confirmar petición del usuario	Ventana para confirmar petición del usuario	
43	Proceso de transformación XSL	Abrir ventana emergente del browser con el contenido del código generado	Ventana emergente con el código fuente java de la interfaz	Mensaje de error por permisos denegados al código Javascript	Error de permisos para abrir el archivo ubicado en el servidor

Tabla 14: Algunas pruebas unitarias del lado del cliente en la Iteración 4

3.7 Iteración 5

- **Planificación**

***Iteración* 5**	
Objetivo	Permitir el almacenamiento de la interfaz en la cuenta del usuario y descargar del código fuente generado. Además implementar el menú que guíe al usuario para listar, modificar y eliminar las interfaces que posea en su cuenta.
Historia de Usuario a Desarrollar	13.- Administrar interfaces almacenadas
	2.- Registro de usuario
	3.- Recuperar contraseña
	4.- Autenticar usuario
	5.- Cambiar contraseña
Tiempo Estimado	11 días
Fecha Inicio/Fin	27-08-2007 / 12-09-2007

Tareas por Historia de Usuario

HU 13.- Administración de interfaces por usuario
- Asociar archivos XML de interfaces a los usuarios.
- Listar interfaces almacenadas.
- Recuperar interfaz almacenada de un archivo físico y cargar componentes en la estructura necesaria para el manejo de construcción de interfaces.
- Dibujar en el área de diseño el diseño solicitado y habilitado para ser modificado.
- Eliminar interfaz almacenada.

HU 2.- Registro de usuario
- Solicitar y validar datos personales del usuario.
- Almacenar usuario en la base de datos del sistema.
- Crear estructura de directorios en el servidor para almacenar los archivos relacionados a las GUI del usuario.

HU 3.- Recuperar contraseña
- Implementar recuperación de contraseña mediante una pregunta de seguridad.

HU 4.- Autenticar usuario
- Verificar existencia del usuario en el sistema para darle acceso.
- Mantener cierta información del usuario en sesión para su verificación en todas las secciones del sistema.

HU 5.- Cambiar contraseña
- Solicitar datos y confirmar validez de la contraseña actual.
- Actualizar contraseña en la base de datos del sistema.

- **Diseño**

Se actualizó el diagrama de clases y el modelo de base de datos con las estructuras necesarias para incluir a los usuarios con sus datos y asociarlos a sus interfaces almacenadas en el sistema. Las modificaciones realizadas a los mismos se observan en las figuras 37 y 38.

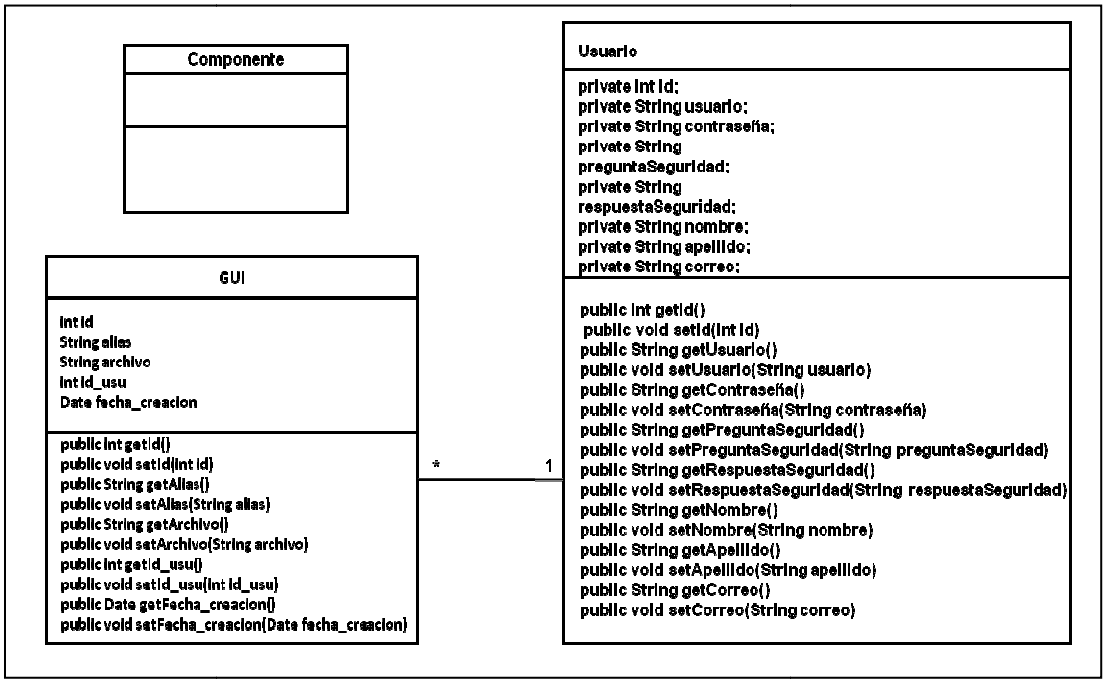


Figura 37: Diagrama de clases. Iteración 5

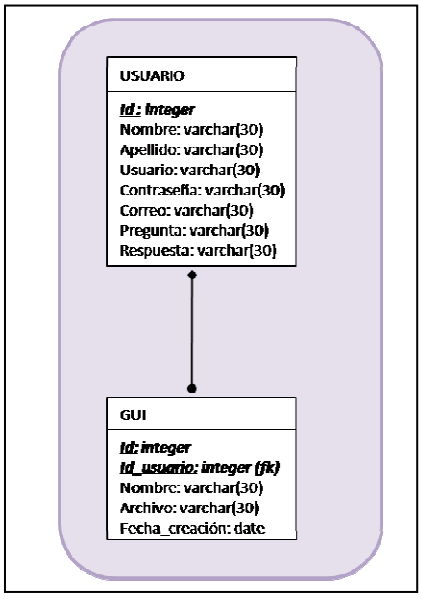


Figura 38: Modelo de Datos. Iteración 5

- **Codificación**

En esta iteración el riesgo se concentró en la recuperación de la interfaz almacenada para ponerla a disposición del usuario en el área de diseño donde se originó. Para hacerlo posible se toma el archivo XML que se generó al momento de recolectar la información para transformarla a código Java y luego de recorrerlo para guardar los componentes en la sesión se envía al cliente; allí utilizando Javascript es procesada y se reutilizan las funciones ya definidas para crear componentes con cada uno de los componentes integrantes del diseño a recuperar. Un fragmento del código que realiza este procesamiento se muestra en la figura 39.

```
function recuperarComponentes(){
..
            /*Variable global que lleva cantidad de componenres actuales*/
            var xmldoc = xml_http.responseXML;
            var componentes = xmldoc.getElementsByTagName('componente')
            count=1;
            //Recorrer las propiedades del componente y actualizar según el valor de cada una
            for( var i=0; i < componentes.length; i++)            {
            //Crear nueva capa para darle formato y convertirlo en el componente a dibujar
            var d = document.createElement("div");
            //Tomar id y tipo del componente
            var id_componente=componentes[i].attributes.getNamedItem("id").value;
            ..
            var tipo_componente=componentes[i].attributes.getNamedItem("tipo").value;
            //Establecer propiedades de cada componente
            d.id=id_componente;
            d.style.position="absolute";
            ..
            d.className="drsElement drsMoveHandle "+tipo_componente+"_html";

            //Tomar todas las propiedades
            var propiedades = componentes[i].getElementsByTagName('propiedad');
            for(var j=0; j< propiedades.length; j++)  {
                        recuperarComponente(propiedades[j],d,tipo_componente);
                  }
            //Programar identificación del componente seleccionado y asignar acciones al serlo
            d.onclick=function(e){
                        var tipo=document.getElementById(e.target.getAttribute('id')).className;
            .. );
            }
            //Buscar datos del formulario
            buscarFormulario(tipo);
            //Buscar datos del componente para controlar propiedades de espacio
            obtenerIdComponenteActual(comp_seleccionado,tipo);
            }
            }
            //Agregar el componente al div principal
            f= document.getElementById("limite");
            if(tipo_componente!="window")f.appendChild(d);
            ..
            }
..
}
```

Figura 39: Parte del código Javascript que procesa data XML de una interfaz para recuperarla al área de diseño

Para finalizar las actividades es esta iteración se implementaron las diferentes secciones relacionadas a las actividades comunes de un usuario del sistema donde mantiene una cuenta: registro, inicio de sesión, recuperación de clave, entre otros, haciendo las validaciones correspondientes y manejando la navegación del usuario en las páginas de la aplicación. Los formularios utilizados se muestran en la figura 40.

Figura 40: Vistas de algunas opciones del sistema para administración de datos del usuario

- **Prueba**

Las pruebas unitarias del lado del servidor fueron sobre una clase creada para el manejo de los datos de los usuarios. Con la aplicación y un usuario de prueba se realizaron diferentes actividades: verificar las validaciones, actualizaciones y mensajes de respuesta.

Pruebas más importantes del lado del servidor (métodos por clase)

Clase: ManejadorGUI

Método BuscarInterfaces

- Verificar que retorne como respuesta las interfaces adecuadas para dicho usuario.

Método EliminarInterfaz

- Verificar que elimine el registro de la interfaz de la BD.

Clase: ManejadorComponente

Método ObtenerComponentesXML

- Verificar que retorne un arreglo con los componentes cargados en el archivo.

Clase: ManejadorArchivos

Método BorrarArchivo

- Verificar que elimine el archivo físico indicado del sistema.

Método CrearCarpeta

- Verificar que la carpeta se cree físicamente en el sistema.

Clase: ManejadorUsuario

Método RegistrarUsuario

- Comprobar que el usuario ingresado existe en el sistema.
- Verificar que tenga todos los datos de creación correctos.

Método ObtenerUsuario

- Verificar que no retorne un objeto nulo al intentar obtener un usuario que si existe.
- Verificar que retorne nulo al buscar un usuario que no existe.

Método Autenticar

- Verificar que retorne como respuesta verdadero si el usuario existe.
- Verificar que retorne como respuesta falso si el usuario no existe.

El método *autenticar* valida un usuario y su contraseña para poder ingresar el sistema, en este sentido las pruebas realizadas verifican que ingresando usuarios válidos e inválidos el método responda si existen o no según, sea el caso. Por otro lado el método *comprobarUsuario* recibe como parámetro un nombre de usuario y revisa si existe uno en el sistema con dicho nombre, ya que no pueden haber usuarios repetidos, las pruebas se basaron en verificar que el método generará como salida verdadero o falso si un nombre de usuario está en uso o no en el sistema (ver código asociado a ambos métodos en la figura 41).

```
public void testAutenticar() {

        //Autenticar un usuario que ya existe
        assertTrue(mu.autenticar("sjavier", "123456"));

        //Autenticar un usuario que no existe
        assertFalse(mu.autenticar("javier", "123456"));

        //Autenticar con la contraseña inválida
        assertFalse(mu.autenticar("sjavier", "123456"));
}
```

```
public void testComprobarUsuario() {

        //Que me compruebe si un usuario existe
        assertTrue("No esta en la BD", mu.comprobarUsuario("sjavier"));

        //Que me compruebe si un usuario no existe
        assertFalse("No esta en la BD", mu.comprobarUsuario("cualquiera"));
}
```

Figura 41: Ejemplo de pruebas unitaria con JUnit en la Iteración 5

Del lado del cliente se hicieron repetidas pruebas recuperando diferentes diseños hasta que el procesamiento del XML con los datos de la interfaz se realizara correctamente y los componentes tuvieran las propiedades visuales y espaciales correctas.

Pruebas más importantes del lado del cliente

- Recuperar una GUI a partir de un archivo XML y dibujar el diseño en pantalla.
- Verificar correspondencia de las características *generales* del componente dibujado en relación a sus propiedades.
- Verificar al seleccionar el componente que sus datos almacenados del lado del servidor correspondan con los dibujados y así no modifiquen sus propiedades ni apariencia.
- Editar propiedades de un componente de la interfaz recuperada.
- Agregar un componente a la interfaz recuperada.

Id	Modulo	Descripción de la Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
39	Modulo de Gestión de Interfaces Almacenadas	Recuperar a partir de los datos de un archivo XML identificado con los datos usuario_interfaz y dibujar el diseño en pantalla	Vista del diseño correspondiente al XML almacenado	Componentes del diseño con propiedades equivocadas	Error en el procesamiento del XML
41	Modulo de Gestión de Interfaces Almacenadas	Editar propiedades de un componente de la interfaz recuperada	Correcta actualización de las propiedades del componente al momento de la edición	Correcta actualización de las propiedades del componente al momento de la edición	
43	Modulo de Gestión de Interfaces Almacenadas	Agregar un componente a la interfaz recuperada	Nuevo componente agregado al diseño recuperado y correcta manipulación de sus propiedades	Nuevo componente agregado al diseño recuperado y correcta manipulación de sus propiedades	

Tabla 15: Algunas pruebas unitarias del lado del cliente en la Iteración 5

- **Observaciones**

Durante las pruebas de aceptación con el cliente, surgen nuevos requerimientos:

- Agregar a la listas de componentes que provee el sistema para el diseño de GUIs un nuevo integrante muy utilizado, el campo para contraseñas.
- Incluir la opción de seleccionar el lenguaje en el que se generará la interfaz gráfica mientras se esté realizando el diseño.

A partir de ellos requerimiento se crearon nuevas historias de usuario y una última iteración:

Número: 16	*Nombre:* Incluir nuevo componente gráfico	
Prioridad: Baja	*Riesgo en desarrollo:* Bajo	*Estimación del tiempo de implementación:* 1 día
Descripción: Agregar a la lista de componentes disponibles de la sección de diseño un campo especial para contraseñas.		

Número: 17	*Nombre:* Selección de lenguaje de código fuente a generar	
Prioridad: Media	*Riesgo en desarrollo:* Medio	*Estimación del tiempo de implementación:* 5 día
Descripción: Incluir en la sección de diseño la opción de escoger que lenguaje se utilizará para generar el diseño en construcción.		

3.8 Iteración 6

- **Planificación**

Iteración 6	
Descripción	Inclusión de un nuevo componente gráfico
Historia de Usuario a Desarrollar	16.- Incluir nuevo componente gráfico.
	17.- Selección de lenguaje de código fuente a generar
Tiempo Estimado	6 día
Fecha Inicio/Fin	20-09-2007 / 21-09-2007

Tareas por Historia de Usuario

HU 16.- Incluir nuevo componente gráfico

- Diseñar versión HTML del nuevo componente.
- Insertar en la base de datos las características iniciales, propiedades, eventos y demás información relacionada con el componente.

HU 17.- Selección de lenguaje de código fuente a generar

- Agregar estructura de datos necesaria para el manejo de componentes y archivos XSL por cada lenguaje a agregar como opción para generar interfaces gráficas.
- Agregar a la sección de diseño un campo para selección de lenguaje para el diseño.
- Implementar un procedimiento para presentar en la barra de herramientas los componentes disponibles para el lenguaje seleccionado.

- **Diseño**

Para incluir un nuevo componente y que éste sea manejado correctamente por el sistema, se investigó primero la clase Swing del componente para conocer sus características visuales, propiedades configurables y eventos, los resultados se muestran en la tabla 16.

Componente	Descripción	Propiedades	Eventos asociados
PasswordField Clase Swing JPasswordField Representación **********	Espacio especial para entrada de contraseñas (sólo una línea disponible, oculta cada carácter sustituyéndolo por un carácter especial específico – carácter de eco).	• Nombre • Valor • Tamaño • Posición • Color • Tamaño Letra • Estilo Letra • Fuente Letra • Color Letra • Carácter de eco	MouseEvent FocusEvent KeyEvent

Tabla 16: Información del componente JPasswordField

Por otra parte, para incluir la funcionalidad de selección de lenguaje para generar la interfaz gráfica se agregaron al modelo de datos del sistema nuevas estructuras para almacenar y relacionar componentes y datos necesarios para la generación a cada lenguaje que se desee incluir como opción de generación de código fuente. Esta expansión se muestra en la figura 42.

También se generó un formato de datos XML (ver figura 43) para llevar los datos de los componentes que deberán aparecer en la barra de herramientas cuando se seleccione alguno de los lenguajes en cuestión del servidor al cliente cuando esta selección se realice. Es importante destacar que se tomó la decisión de eliminar del diseño aquellos componentes que no estén disponibles en el lenguaje seleccionado por el usuario.

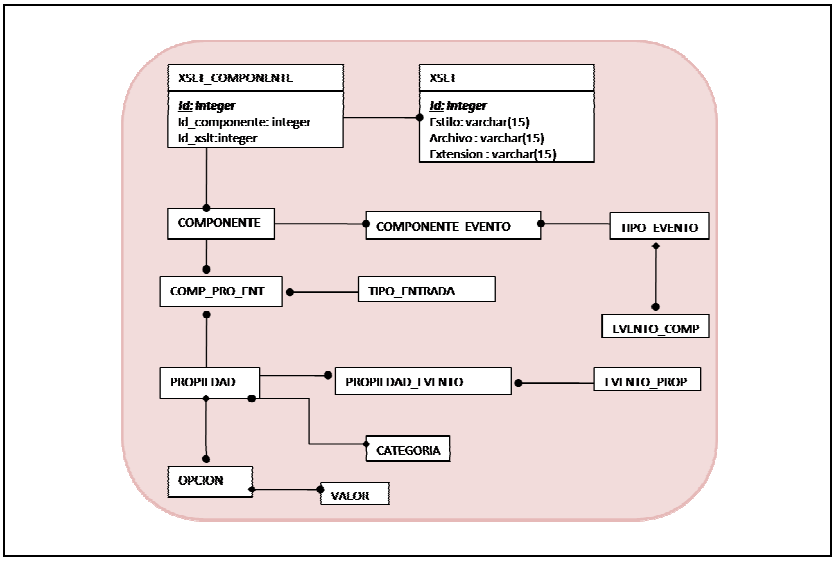


Figura 42: Modelo de Datos. Iteración 6

<componentes> <componente id=" tipo="> <tooltip></tootip> <texto></texto> </componente> </componentes>	<!ELEMENT componentes (componente+)> <!ELEMENT componente (tooltip, texto)> <!ATTLIST componente id CDATA #REQUIRED> <!ATTLIST componente tipo CDATA #REQUIRED> <!ELEMENT text #PCDATA> <!ELEMENT tooltip #PCDATA>

Figura 43: Estructura XML de datos para colocar en la barra de herramientas los componentes disponibles en el lenguaje seleccionado y su DTD correspondiente

- **Codificación**

En la etapa de codificación, del lado del servidor se agregó un nuevo atributo a la clase componente (carácter de eco) y los métodos necesarios para asignar y consultar su valor. Además se modificó el proceso encargado de creación de XML de cualquier componente gráfico para incluir la presencia de la nueva propiedad. Del lado del cliente se modificó el proceso encargado de manejar la asignación de valores a las propiedades de un componente, para añadir al carácter especial para el campo de contraseña y por último establecer las restricciones de tamaño para el nuevo componente.

```
..
switch (tipo_componente_actual){
        case "window":
        //Limites de altura
        if (count>1  && mayorY()>100){
                minHeight=parseInt(mayorY());
        }else{
                minHeight=100;
        }
        if (count>1 && mayorX()>150){
                minWidth=parseInt(mayorX());
        }else{
                minWidth=150;
        }
        habilitar_ancho=0;
        break;

        case "button":
        //Limites de altura
        minHeight=13; minWidth=40;habilitar_ancho=0;
        break;
        ..
         //Restriccion del tamaño del campo para password
        case "password":
        //Limites de altura
        minHeight=18; minWidth=20; maxdY=0;habilitar_ancho=1;
        break;
        ..
}
```

Figura 44: Parte del código Javascript encargado de controlar restricciones de tamaño de los componentes

Con respecto a la implementación de la funcionalidad para seleccionar el lenguaje en el cual se generaría el código fuente de la interfaz, se creó un procedimiento para lograr que los componentes que aparecen en la barra de herramientas, en el área de diseño y el documento XSL a utilizar para hacer la generación de código dependan del lenguaje que seleccione el usuario.

Cada vez que el usuario cambia la opción del lenguaje a utilizar para generar la interfaz, se actualizan los componentes presentes en la barra de herramientas a

partir de la información proveniente de la base de datos del sistema donde se relaciona cada lenguaje con los componentes que soporta.

Al momento de guardar y/o generar la interfaz se guarda en el archivo XML presentado en la iteración 3 el identificador del lenguaje en el que será generado el código fuente. Este dato se utiliza también al momento que el usuario recupera su interfaz, así se dibuja la interfaz, se coloca como seleccionado el lenguaje en el que se guardó la última vez y se presentan en la barra los componentes disponibles para continuar o modificar el diseño. Para realizar las pruebas correspondientes a esta funcionalidad, se agregó como opción para generar interfaces gráficas: "Java (Swing Básico)", para el cual sólo definimos 3 componentes disponibles: botón, etiqueta y cuadro de texto.

```
<gui nombre_clase=" lenguaje="">
   <componentes>
      <componente id="  tipo="">
       ..
       ..
      </componente>
   </componentes>
       ..
       ..
</gui>
```

```
<!ELEMENT gui (componentes, complementos?)>
<!ATTLIST gui nombre_clase CDATA #REQUIRED>
<!ATTLIST gui lenguaje CDATA #REQUIRED>
<!ELEMENT componentes (componente+)>
<!ELEMENT componente (propiedades,eventos?)>
<!ATTLIST componente id CDATA #REQUIRED>
<!ATTLIST componente tipo CDATA #REQUIRED>
 ..
 ..
```

Figura 45: Extracto modificado de la estructura XML con datos de la interfaz gráfica y su DTD correspondiente

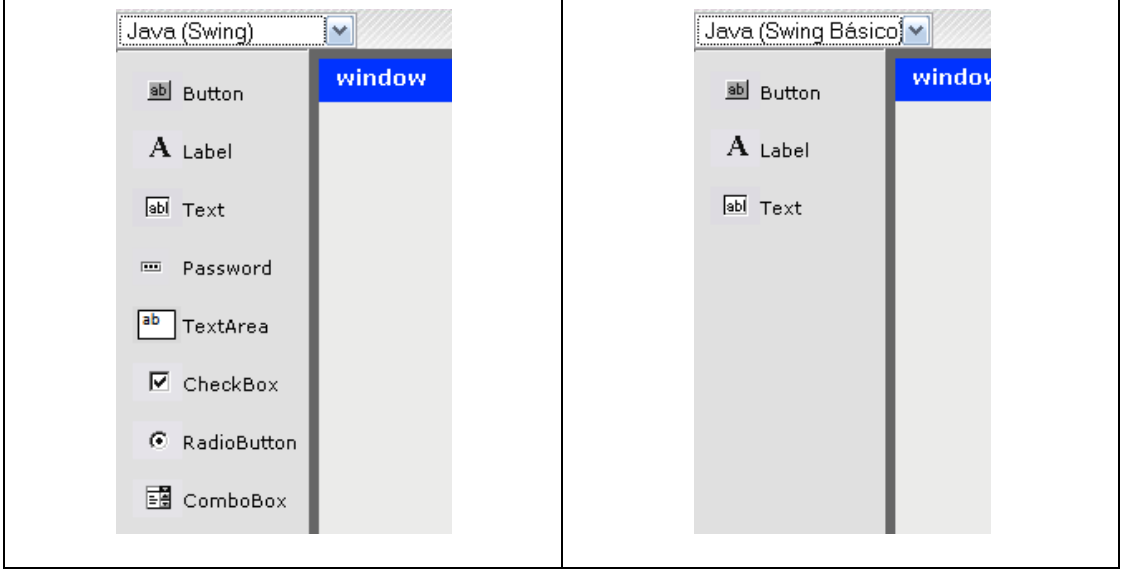
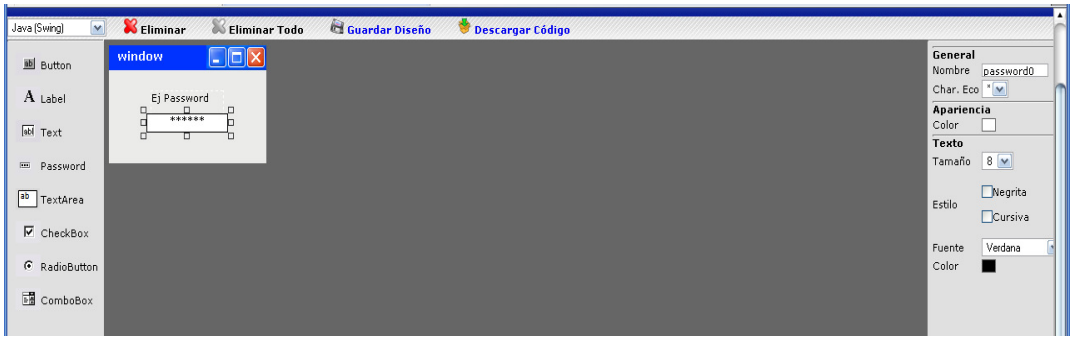


Figura 46: Ejemplo de 2 barras de herramientas diferentes dependiendo de las características del lenguaje seleccionado

- **Pruebas**

Se realizaron un conjunto de pruebas realizando diseños de interfaces que incluían al nuevo componente para verificar el manejo de sus propiedades y la generación de código fuente Java correspondiente para la creación del mismo. Una de las vistas y resultados de estas pruebas se puede apreciar en la figura 43.




```
/*
*prueba_pass.java
* Fecha:
* Autor
* Version:
*/

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class prueba_pass extends JFrame{
..
..
private JPasswordField password0;
..
password0.setBounds(24,70,92,18);
..
//Caracter de eco para el passwordField
password0.setEchoChar('?');
..
```

Figura 47: Resultados obtenidos al incluir un campo de contraseña en una interfaz (diseño y código generado)

Además, esos procesos de diseño se cambió el valor del lenguaje seleccionado en repetidas ocasiones para verificar que se mostraran los componentes correspondientes y se eliminaran del diseño aquellos que no fueran soportado por el lenguaje seleccionado.

Específicamente, las pruebas más importantes del lado del cliente fueron las siguientes:

- Cambio de lenguaje de Java (Swing) a Java (Swing Basico): Verificar barra de componentes.
- Cambio de lenguaje de Java (Swing Básico) a Java (Swing): Verificar barra de componentes.
- Cambio de lenguaje de Java (Swing) a Java (Swing Basico) con un TextArea en el diseño: Verificar eliminación de componentes no disponibles.
- Cambio de lenguaje de Java (Swing) a Java (Swing Basico): Verficar tooltips en cada componente de la barra.
- Guardar diseño de interfaz en Java (Swing Básico) y luego recuperarla.

Id	Modulo	Descripción de la Prueba	Resultado Esperado	Resultado Obtenido	Motivo de la Falla
54	Manejo de componentes gráficos	Cambio de lenguaje de Java (Swing) a Java (Swing Básico): **Verificar barra de componentes**	En la barra de componentes solo aparecerán: botón, etiqueta y campo de texto	En la barra de componentes solo aparecen: botón, etiqueta y campo de texto),pero no aparecen los íconos	Estilo CSS de los componentes de la barra debe ser dinámico
56	Manejo de componentes gráficos	Cambio de lenguaje de Java (Swing) a Java (Swing Básico) con TextArea en el diseño**: Verificar eliminación de componentes no disponibles**	Diseño sin componentes	Diseño sin componentes	
57	Manejo de componentes gráficos	Cambio de lenguaje de Java (Swing) a Java (Swing Basico): Verificar tooltips en cada componente de la barra	Tootips correctos en cada componente	Tootips correctos en cada componente	
59	Manejo de componentes gráficos	Guardar diseño de interfaz en Java (Swing Básico) y luego recuperarla	Barra de componentes de Java Swing Básico (solo aparecerán: botón, etiqueta y campo de texto)	Barra de componentes de Java Swing Básico (solo aparecerán: botón, etiqueta y campo de texto)	

Tabla 17: Algunas pruebas unitarias del lado del cliente en la Iteración 6

Seguidamente se realizaron pruebas integrales sobre el sistema haciendo un recorrido por todas las funcionalidades del mismo, estableciendo un escenario que incluyó el registro de un usuario nuevo en el sistema, diseño una GUI, generación de su código y luego la recuperación la misma, edición y generación del nuevo código fuente.

Además, se dedicaron unos días para refactorizar código y agregar o modificar organización de algunos procedimientos para optimizarlo (tanto el código Javascript como las clases Java): unificar mensajes de error al usuario, agregar manejo de excepciones en todos los métodos, dividir procedimientos y hacerlos mas modulares, entre otras actividades.

3.9 Resumen del capítulo

En este capítulo se logró apreciar la puesta en práctica de una personalización del proceso de desarrollo XP que incluyó las 4 fases básicas de desarrollo de software (análisis de requerimientos, diseños, implementación y prueba) en iteraciones, previamente planificadas a partir de variables determinantes como prioridad por funcionalidad, riesgo y tiempo de desarrollo. Además de una fase extra que tuvo lugar en aquellas iteraciones en las que la intervención del cliente durante las pruebas de aceptación causó modificaciones en la planificación, requerimientos o resultados de la iteración en cuestión o en iteraciones siguientes. Todo lo anterior permitió una implementación organizada, fluida y exitosa del sistema.

Las pruebas jugaron un papel muy importante en el desarrollo del sistema y por lo tanto fueron registradas cuidadosamente y bien documentadas, al igual que se elaboraron aquellos modelos que el equipo de desarrollo consideró necesarios de la manera más sencilla en cada iteración, siguiendo los principios de Modelación Agil.

Durante las pruebas de aceptación de la quinta iteración (que se había planificado como última), se presentaron nuevos requerimiento que pudieron cubrirse eficientemente gracias a las buenas prácticas de codificación, documentación y comunicación entre los programadores durante todo el proceso de desarrollo.

Finalmente el desarrollo se realizó en 6 iteraciones, creando una aplicación Web de nombre CELIG: *Constructor en Línea de Interfaces Gráficas*, capaz de manejar componentes gráficos y configurar sus propiedades para diseñar una GUI de aplicaciones de escritorio y generar satisfactoriamente su código fuente Java (usando las interfaces de la librería Swing), almacenarla y ponerla a disposición del usuario para editarla y generarla de nuevo cuando lo desee.

CAPITULO IV: ESCENARIOS

En este capítulo se pretende utilizar el sistema CELIG en casos de implementación de interfaces para aplicaciones de escritorio Java en contextos reales. Para ellos se establecieron 2 casos donde fueran necesarias algunas ventanas con un propósito, funcionalidad y componentes específicos. En ambos se utilizaron interfaces con componentes de la librería Swing.

4.1 Descripción de Escenarios:

4.1.1 Escenario 1: Básico a Medio

El primer caso incluye un diseño de GUI sencillo (nivel Básico), una ventana pequeña con un componente de entrada de datos, una etiqueta que indique que se solicita la cédula de identidad al usuario para poder tener acceso a un sistema y dos botones para entrar o para cancelar la acción.

Luego, para aprovechar la recuperación de GUIs para su edición y generación del código fuente nuevo, asumiremos que se presenta una modificación del requerimiento, una verificación extra para mayor seguridad. Para esta nueva autenticación se debe especificar una clave propia del usuario (a este nuevo diseño lo clasificamos como nivel Medio).

4.1.1.1 Prototipo de interfaz a diseñar:

En las figuras 44 y 45 se pueden observan los modelos de interfaz gráfica a construir (el modelo inicial y su siguiente versión con más datos).

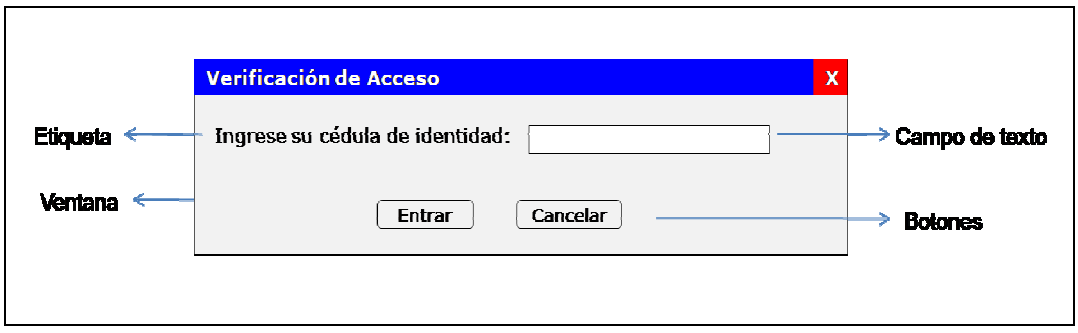


Figura 48: Prototipo de GUI. Diseño básico

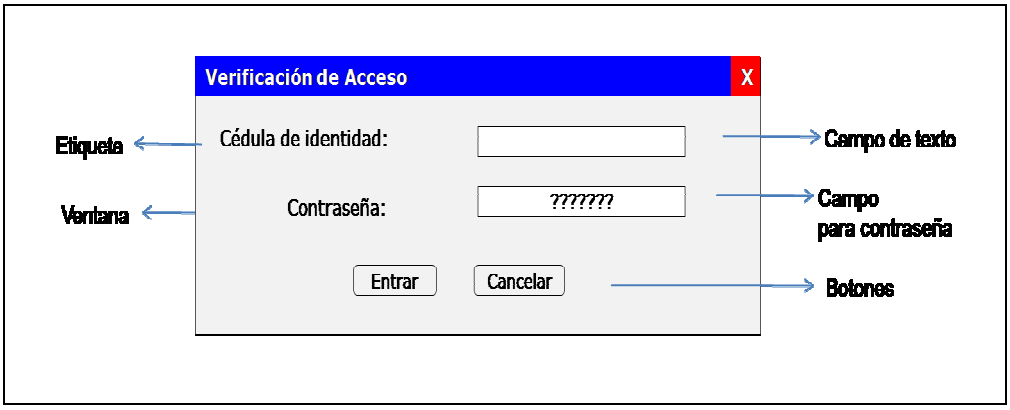


Figura 49: Prototipo de GUI. Diseño medio

4.1.1.2 Procedimiento de creación de GUI con CELIG

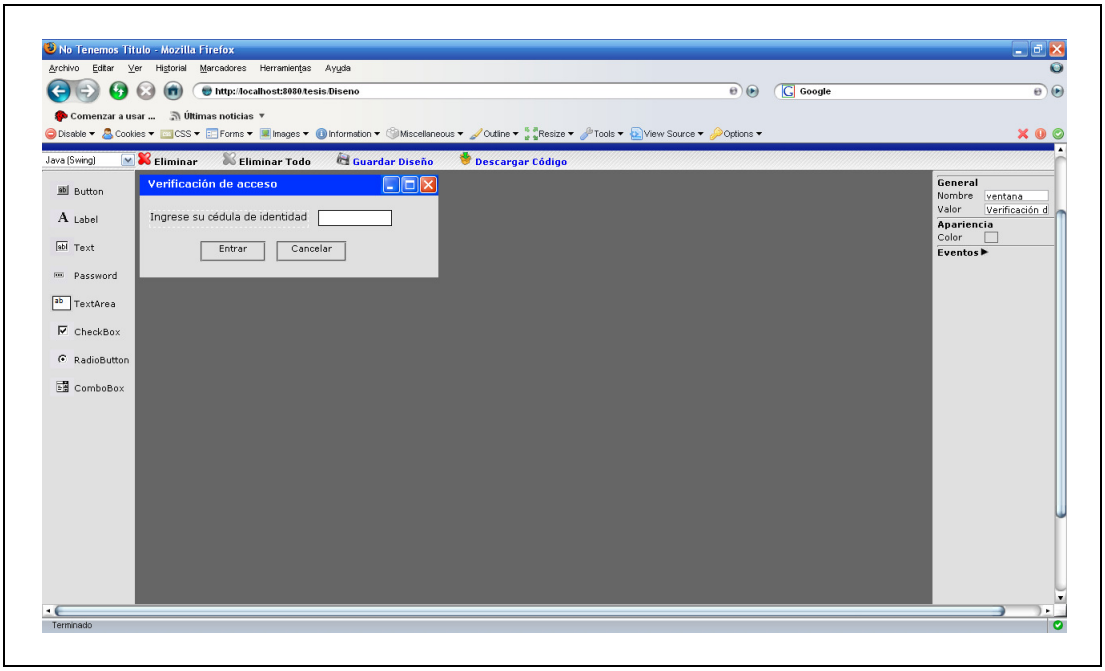


Figura 50: Vista del sistema al finalizar el diseño (Diseño básico)

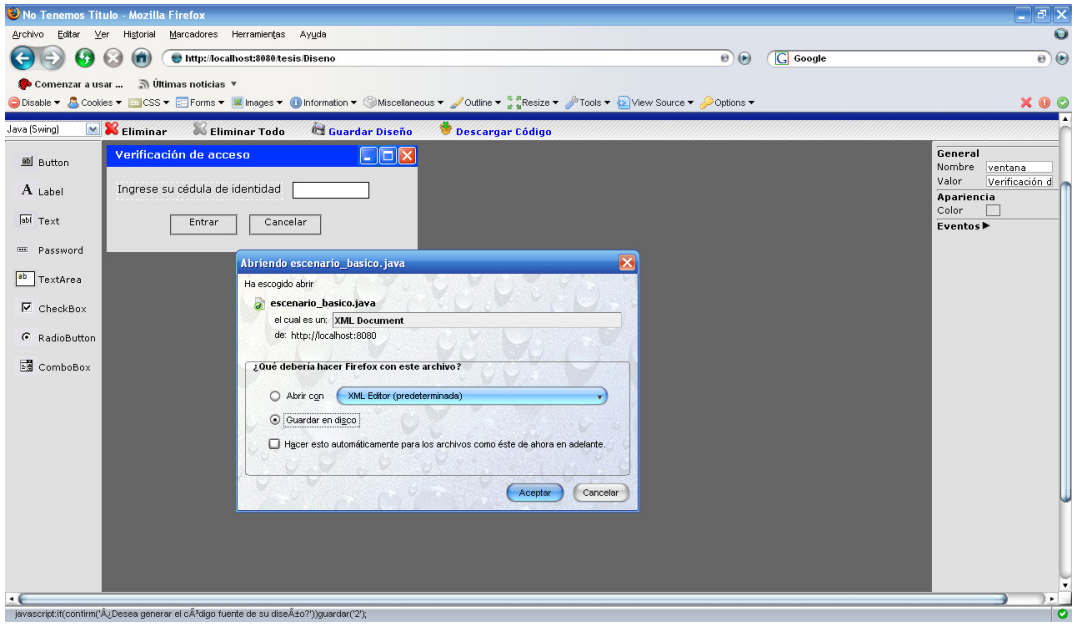


Figura 51: Vista del momento en que el usuario obtiene el código fuente Java (Diseño básico)

```
/*
*escenario_basico.java
* Fecha:
* ../
import javax.swing.*;
..
public class escenario_basico extends JFrame{

//Declaracion del marco principal de la ventana
private JFrame ventana = new JFrame("Verificación de acceso");

//Declaracion de variables represetantes de los componentes de la ventana
private JLabel label0;
..
private JPanel contentPane;

public void JVentana() {
//Inicialicacion de componentes con su valor
label0 = new JLabel("Ingrese su cédula de identidad");
..
button0 = new JButton("Entrar");
contentPane = (JPanel)this.getContentPane();

//Establecer posicion y dimension de cada componente (x,y,ancho,alto)
label0.setBounds(10,20,197,20);
..
 //Establecer color de fondo de los componentes
label0.setOpaque(true);
label0.setBackground(new Color(0Xe0e0e0));
..
text0.setFont(new Font("Verdana",Font.PLAIN,8));
..
button0.setForeground(new Color(0X000000));
//Agregar componentes al marco principal de la ventana
contentPane.setLayout(null);
contentPane.add(label0);
..
//Estableces panel de contenido de la ventana principal
ventana.add(contentPane);
ventana.setSize(373,128); ..
```

Figura 52: Partes del código fuente Java generado por CELIG. (Diseño básico)

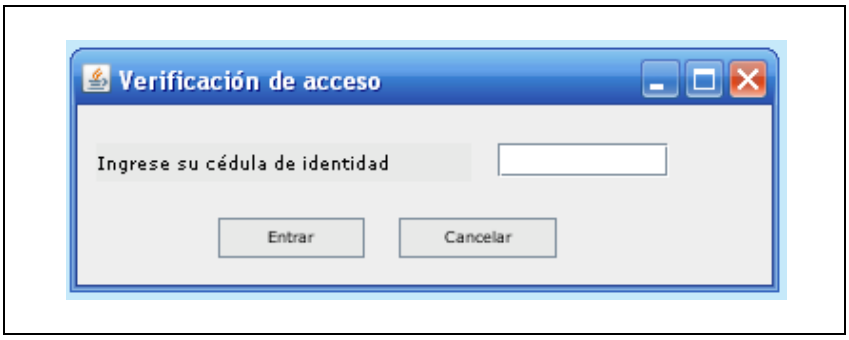


Figura 53: Vista del resultado de ejecutar código fuente Java generado por CELIG (Diseño básico)

Luego el usuario puede acceder al área de Interfaces y consultar aquellas que ha almacenado anteriormente. Allí encontrará el diseño del primer escenario.

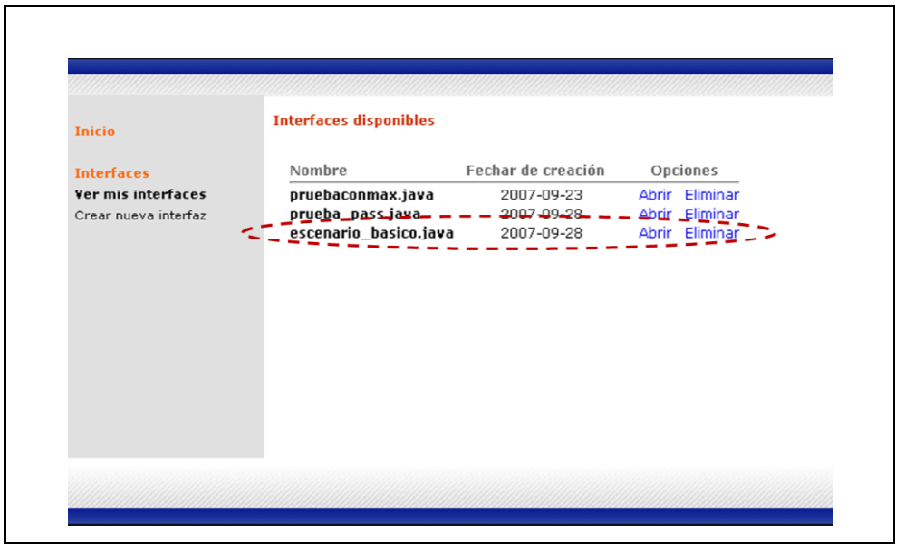


Figura 54: Vista del sistema al listar interfaces disponibles para el usuario

Al seleccionarlo para abrir, el sistema se dirige automáticamente al área de diseño y muestra la interfaz seleccionada lista para modificar y/o generar nuevamente.

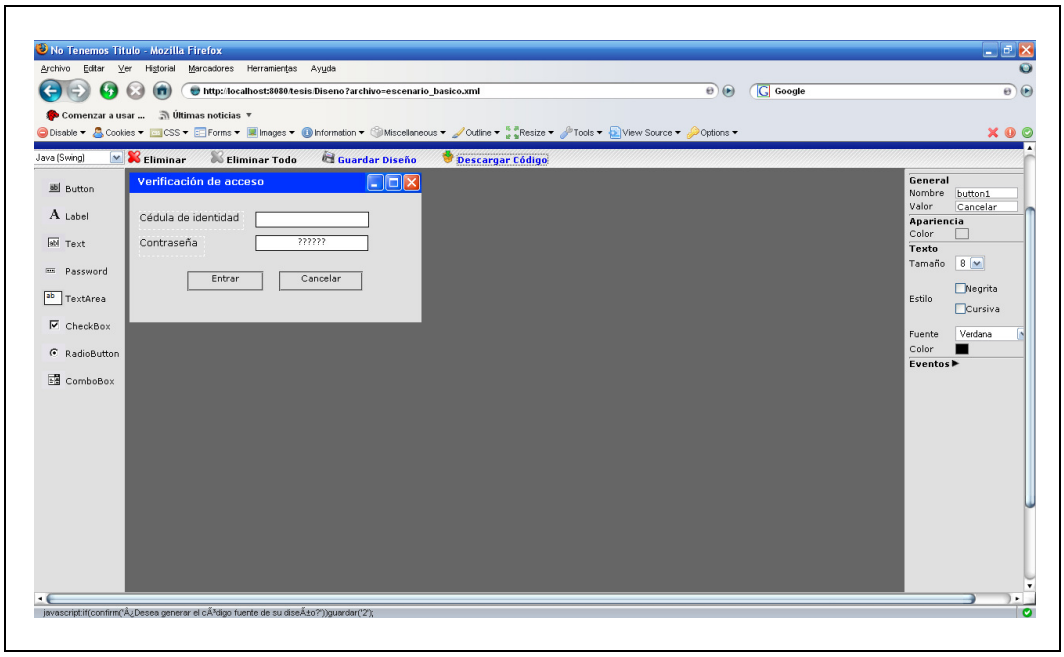


Figura 55: Vista de la interfaz modificada. (Diseño medio)

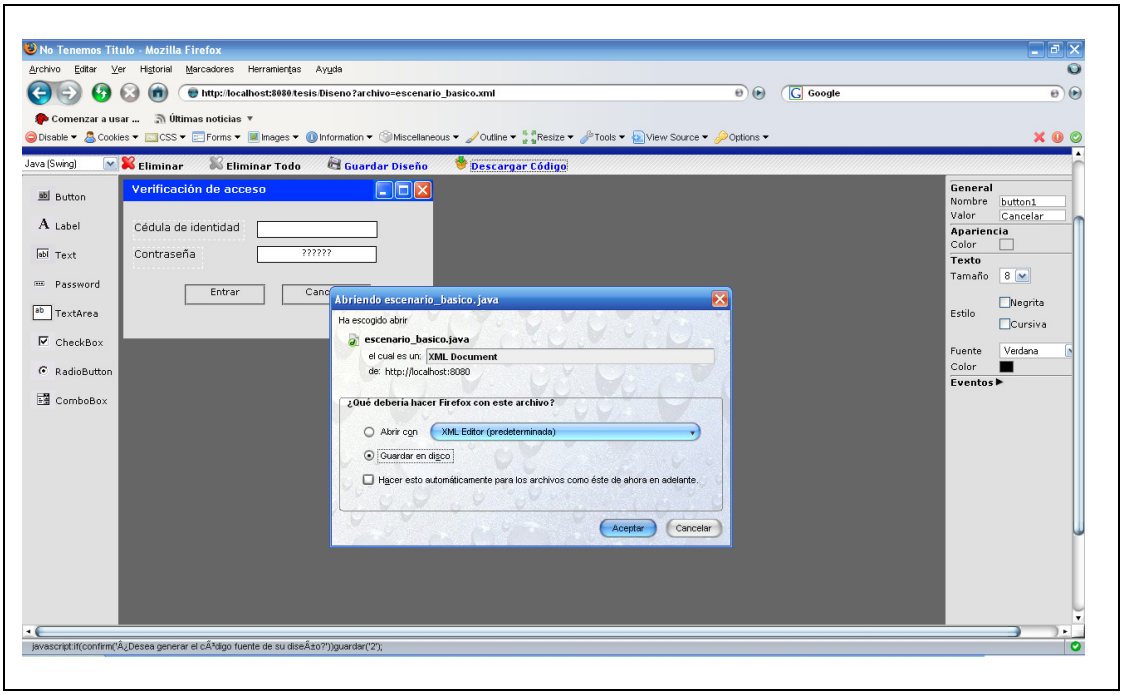


Figura 56: Vista del momento en que el usuario obtiene el código fuente Java. (Diseño medio)

```
/*
*escenario_basico.java
* Fecha:
*../
..
//Declaracion de variables represetantes de los componentes de la ventana
private JLabel label0;
..
private JPasswordField password0;
private JPanel contentPane;

public void JVentana() {
//Inicialicacion de componentes con su valor
label0 = new JLabel("Cédula de identidad");text0 = new JTextField("");
..
password0 = new JPasswordField();
contentPane = (JPanel)this.getContentPane();

//Establecer posicion y dimension de cada componente (x,y,ancho,alto)
..
password0.setBounds(159,56,142,18);

//Establecer color de fondo de los componentes
..
password0.setBackground(new Color(0Xffffff));

//Propiedades de texto, tipo de letra, estilo de la letra y tamano
label0.setFont(new Font("Verdana",Font.PLAIN,9));
..
password0.setFont(new Font("Verdana",Font.PLAIN,8));

//Establecer color de la fuente de los componentes
..
password0.setForeground(new Color(0X000000));

//Caracter de eco para el passwordField
password0.setEchoChar('?');
..

//Estableces panel de contenido de la ventana principal
ventana.add(contentPane);
..
//Ultimo paso, hacerla  visible
ventana.setVisible(true);
}
}
```

Figura 57: Partes del código fuente Java generado por CELIG. (Diseño medio)

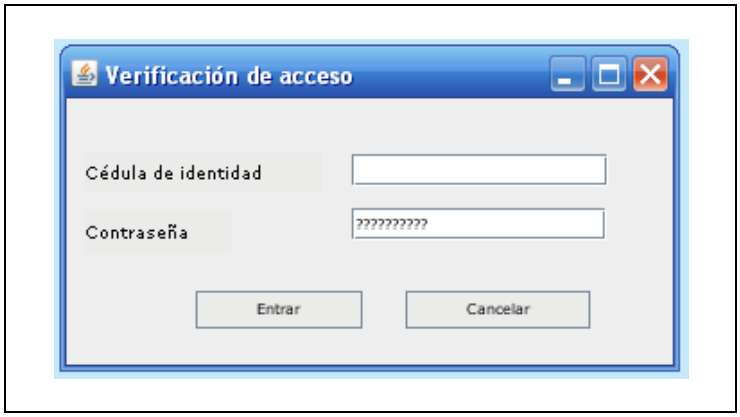


Figura 58: Vista del resultado de ejecutar código fuente Java generado por CELIG (Diseño medio)

4.1.2 Escenario 1: Avanzado

En este escenario, se necesita una ventana donde se puedan tener diferentes componentes gráficos capaces de tomar los datos necesarios para elaborar un recibo de pago. Esta ventana tiene otras especificaciones en cuanto a tipos de letra y colores que se observan en el prototipo de la figura 56.

4.1.2.1 Prototipo de interfaz a diseñar:

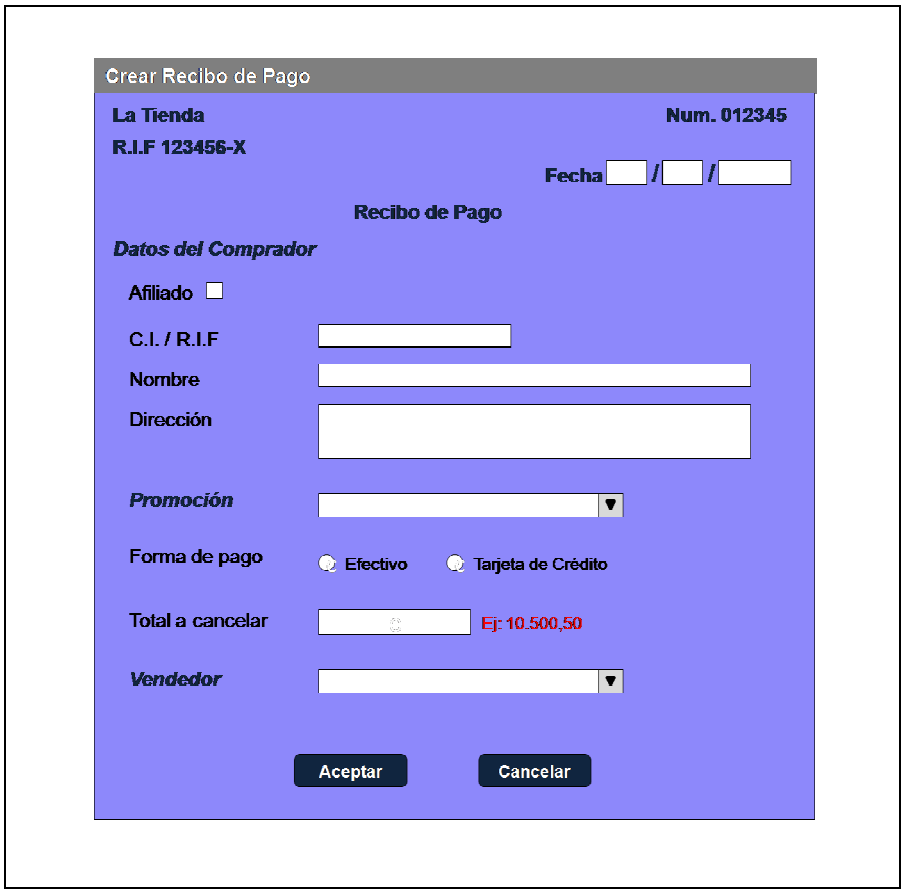


Figura 59: Prototipo de GUI. Diseño avanzado

4.1.2.2 Procedimiento de creación de GUI con CELIG

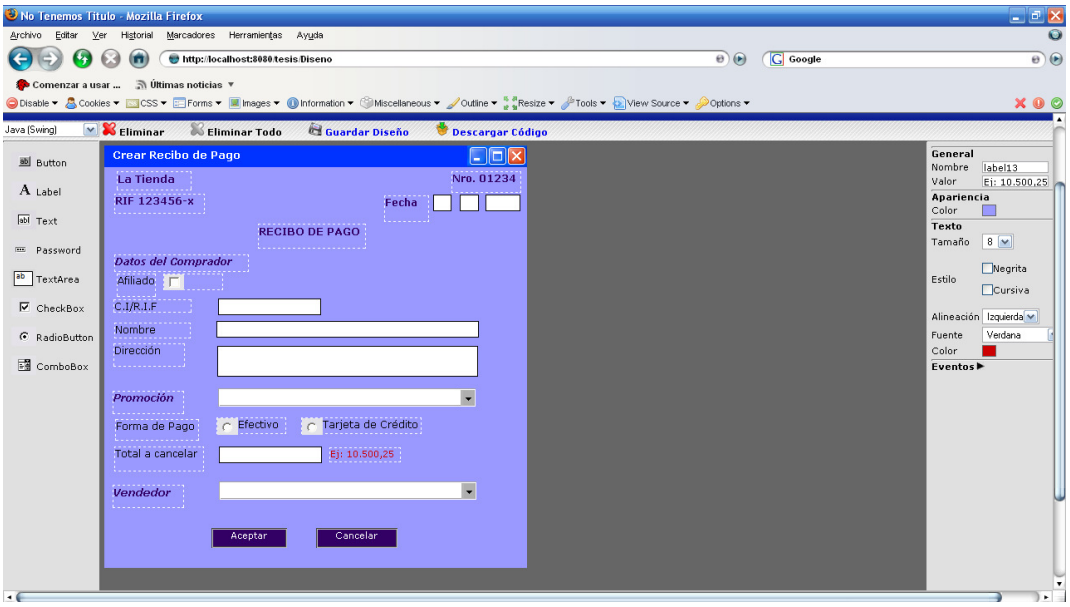


Figura 60: Vista del sistema al finalizar el diseño (Diseño avanzado)

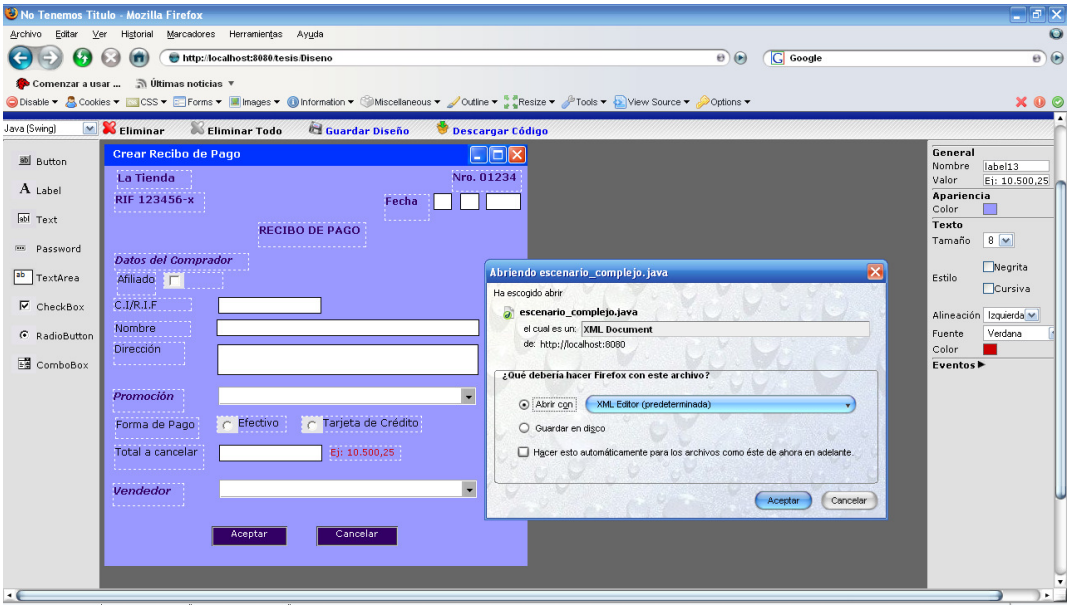


Figura 61: Vista del momento en que el usuario obtiene el código fuente Java (Diseño avanzado)

```
/*
*escenario_complejo.java
import javax.swing.*;
..
//Declaracion del marco principal de la ventana
private JFrame ventana = new JFrame("Crear Recibo de Pago");
//Declaracion de variables represetantes de los componentes de la ventana
private JButton button1; String valores_combo1[] = {"Jose","Maria"};
private JComboBox combo1;
private JButton button0;
private JLabel label14;
private JLabel label13;
private JTextField text5; String valores_combo0[] = {"2x1","3x2"};
private JComboBox combo0;
..
button1 = new
JButton("Cancelar");
combo1 = new JComboBox(valores_combo1);
button0 = new JButton("Aceptar");
label14 = new JLabel("Vendedor");label13 = new JLabel("Ej: 10.500,25");text5 = new JTextField("");
combo0 = new JComboBox(valores_combo0);
..
ButtonGroup g1 = new ButtonGroup();
g1.add(radio0);
g1.add(radio1);
..
//Color de fondo de la ventana
contentPane.setBackground(new Color(0x9999ff));
//Ultimo paso, hacerla  visible
ventana.setVisible(true);
}
}
```

Figura 62: Partes del código fuente Java generado por CELIG. (Diseño avanzado)

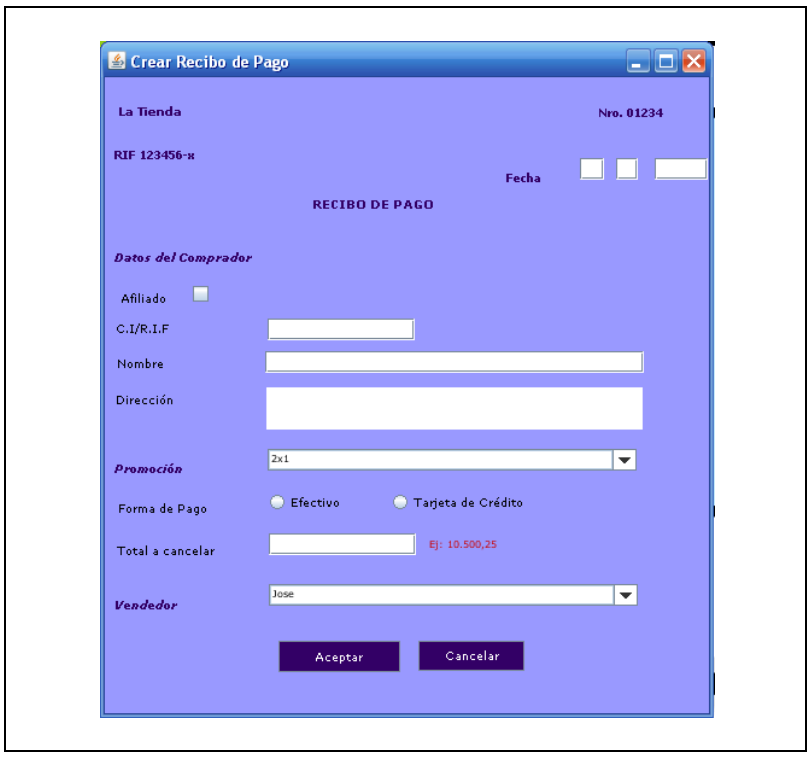


Figura 63: Vista del resultado de ejecutar código fuente Java generado por CELIG (Diseño avanzado)

Como se pudo observar en la serie de figuras que muestran paso a paso el proceso de creación de cada prototipo con la aplicación, las pruebas resultaron exitosas; además durante dicho proceso el sistema respondió satisfactoriamente a pequeñas verificaciones como: reporte de errores con nombres de archivos incorrectos, valores incorrectos en la propiedades de componentes, actividades prohibidas (eliminar la ventana), entre otras, manteniéndose robusta y evitando propagación de errores del usuario que puedan causar problemas con el código fuente resultante .

RESULTADOS Y CONCLUSIONES

Culminada la realización de este Trabajo Especial de Grado, atribuimos en gran parte el logro de los objetivos planteados al principio de éste documento a la planificación y organización de requerimientos y actividades necesarias para crear CELIG, el cual cumple con todos los objetivos propuestos en la solución del problema identificado en el Capítulo I.

La investigación y desarrollo realizados dio como resultado una nueva herramienta de generación de código fuente de interfaces gráficas de aplicaciones de escritorio, de gran utilidad para los usuarios programadores.

La implementación del diseño de GUIs no fue sencilla. Darle al usuario el control de componentes de la página, que en este caso son componentes con características y restricciones específicas para cada uno, implica la validación de una gran cantidad de casos, combinaciones de eventos y pasos, para detectar posibles errores y manejarlos para evitar que el usuario se encuentre con detalles que puedan entorpecer su actividad en la página, el correcto funcionamiento de la aplicación y/o los resultados generados.

Sin embargo, se logró el manejo de eventos vinculados a la actividad del usuario así como también proveer una interfaz interactiva que permite al usuario manipular componentes gráficos y sus propiedades de manera rápida, sencilla y ver los cambios de sus acciones en tiempo real, todo esto dentro del ámbito de una aplicación Web, que hizo que las historias de usuarios con actividades asociadas fueran de alto riesgo en implementación.

Con el fin de presentar las características de nuestra aplicación como nueva integrante del grupo de herramientas de generación de interfaces existentes, se muestra a continuación el resultado de evaluar al sistema CELIG con los mismos criterios utilizados en el estudio de algunas de estas aplicaciones en el capítulo 2, en la sección 2.2.3.3:

Usabilidad de la herramienta	
¿La interfaz es simple y sencilla visualmente?	Si
¿Las funciones para la construcción de interfaces aparecen inicialmente?	Si
¿Los controles de la herramienta para manipular los componentes están claramente visibles y sus funciones son identificables?	Si
¿Minimiza las oportunidades para que el usuario cometa errores?	Si
Funcionalidad de la herramienta	
¿Se puede cambiar el tamaño de los componentes?	Si
¿Se puede cambiar el color de los componentes?	Si
¿Se puede cambiar las propiedades de los textos de la interfaz?	Si
¿El código fuente asociado a la ventana se a generando en paralelo al diseño de ésta?	No
Características del código fuente generado	
¿El archivo java generado incluye el encabezado con información sobre la clase?	Si
¿El archivo java generado tiene documentación sobre los métodos y variables utilizadas?	Si
Características de la interfaz generada	
¿Aparecen todos los componentes utilizados en el diseño de la ventana?	Si
¿La distribución de los componentes corresponde al diseño?	Si
¿El tamaño de los componentes es el acorde a su contenido?	Si
¿El color de los componentes es el especificado en el diseño?	Si
¿El formato de la fuente es el especificado en el diseño?	Si

Como se puede observar, uno de los criterios de funcionalidad no se cumple, sin embargo es una característica que no afecta ni limita las actividades del usuario ni la calidad del diseño o código generado por la aplicación. Además, podemos incluir como ventaja un aspecto que no se refleja en los criterios: CELIG por ser una aplicación Web, no requiere instalación, no tiene requerimientos técnicos especiales para poder funcionar y puede ser utilizada desde donde y cuando se necesite sin limitaciones.

Otra característica que importante destacar es la capacidad de expansión de CELIG, ya que su diseño y estructura se definieron con el fin de generar interfaces gráficas para aplicaciones de escritorio en diferentes librerías o lenguajes de programación. Para esto son pieza clave el manejo de datos con XML generales para cualquier componente y la transformación XSL que genera el código fuente a partir de esa data.

Ahora bien, durante todo el proceso de creación de CELIG se involucraron factores de investigación, aprendizaje y análisis que fueron guía en las actividades

requeridas para el cumplimiento de cada objetivo. Estos se presentan a continuación:

La personalización XP para nuestro proyecto fue clave para el éxito del mismo, gracias a la aplicación de prácticas del proceso en diseño (pudiendo incluir parte de Modelación Agil), codificación y pruebas que soportaron el desarrollo rápido de una aplicación expuesta a cambios durante las etapas de su creación y sobretodo promoviendo la comunicación y trabajo en equipo de los programadores, comprometidos en cumplir sus labores en cada iteración. Además, trabajar con los factores de prioridad, riesgo y tiempo determinó la toma de decisiones al momento de planificar que crearon un buen ritmo de implementación y solución de requerimientos funcionales y no funcionales en el momento indicado.

Por otra parte, es la primera vez que intentamos llevar a la Web una aplicación existente sólo en herramientas de escritorio, esto nos llevó a explorar ejemplos de éstas, aprender a utilizarlas, investigar sobre los conceptos que manejan, su funcionamiento y resultados, lo cual implica un aprendizaje importante sobre las diferentes formas que se emplean para manejar la interacción de los usuarios con estas aplicaciones y como poder simularlas en una plataforma diferente.

Finalmente, una gran experiencia fue haber aprendido y aplicado en este proyecto las tecnologías que utilizamos para desarrollar módulos de la aplicación, en especial del lado del cliente con tecnologías que trabajando independientemente (como DOM, Javascript, XML) y en conjunto (AJAX) fueron la base para lograr la interactividad y dinamismo en el contenido de la página. AJAX fue la solución ideal para lograr la actualización del componente y sus propiedades al instante que el usuario las modificaba; y XML el formato para manejo de data del diseño en construcción que resultó exitoso además para la generación de código fuente Java con las características y estándares planteados al aplicarle una transformación con un archivo XSL independiente. Cabe destacar que las propiedades de XSL fueron tomadas en cuenta para el diseño del XML a transformar debido a sus limitaciones como lenguaje de estilos.

RECOMENDACIONES

CELIG es una aplicación diseñada para ser utilizada por cualquier desarrollador que necesite implementar una interfaz gráfica de usuario para una aplicación de escritorio en lenguaje Java.

A nivel de implementación de dicha aplicación: su estructura, los formatos de XML generados para el intercambio de datos durante el proceso de diseño de la GUI y diseño tanto del lado del cliente por la modulación del manejo de los componentes como del servidor con la transformación XSL dan lugar a la expansión de las funcionalidades de la misma, pudiendo agregar en siguientes versiones:

- Inclusión de componentes gráficos Swing para la interfaz como paneles, listas, imágenes, menús, entre otros.
- Permitir al usuario programar los eventos que asigna a cada componente al momento de diseñar la interfaz (para esta funcionalidad sería importante incluir la validación del código fuente Java que se escriba para mantener la calidad del código generado).
- Opción de presentar una vista preliminar de la ventana en construcción.
- Configuración del look and feel de la interfaz por el usuario, aprovechando que Swing lo permite.
- La opción de construir GUI con librerías AWT y SWT.
- La opción de construir código fuente de GUI en otros lenguajes de programación como C, VisualBasic, Delphi, entre otros.
- Construcción de applets.
- Construcción de formularios y/o interfaces para aplicaciones Web.

Sin embargo, una mejora que podría resultar prioritaria parte del hecho que CELIG no funciona correctamente en Internet Explorer, debido a las diferencias de implementación de Javascript y CSS que requiere cada navegador. Una actividad importante sería expandir en todo el procesamiento implementado en lenguaje Javascript la utilización de los métodos de la librería Prototype (que en esta versión del sistema se utilizaron en el manejo de los componentes en el área de diseño, su actualización, movimiento y cambio de tamaño), cuyas versiones van evolucionando eliminando cada vez más las diferencias entre navegadores.

Igualmente pensamos que la constante expansión de la tecnología y más aún la Web 2.0, harán posible la inclusión de más funcionalidades y mejoras en la presentación e interacción del usuario con las mismas, que promoverán la optimización de herramientas de desarrollo que actualmente están en la Web, así como también la creación de muchas más que simplifiquen las tareas del día a día de usuarios de aplicaciones informáticas de cualquier área; pero es importante que al hacerlo, se mantenga la efectividad de los resultados generados por éstas.

REFERENCIAS BIBLIOGRÁFICAS

[Aguil2002] Introducción a la Programación Extrema
Aguilar Sierra, Alejandro
Universidad Nacional Autónoma de México
Diciembre, 2002

[Beck2000] Planning Extreme Programming
Kent Beck, Martin Fowler
Addison Wesley
Primera Edicion
Octubre 2000

[U-INSW] Ingeniería de Software
Disponible en: http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html
Ultima actualización: Septiembre, 2002

[Ambl2000] Agile Modeling (AM). A Practices-Based Approach For Real-World Development
Scott W. Ambler
Ronin International, 2001

[Marin2002]Swing (I)
Andrés Marín López
Universidad Calos III de Madrid
Departamento de Ingeniería Telemática

[BMSJ2007] Estudio para el desarrollo de una aplicación Web generadora de código fuente Java de interfaces gráficas de usuario para aplicaciones de escritorio
Brianto Madelyx, Sandoval Javier
Seminario en Aplicaciones con Tecnología Internet
Universidad Central de Venezuela.
Abril 2007

Cascading Style Sheets, designing for the Web
Håkon Wium Lie, Bert Bos
Editorial Addison Wesley
Segunda Edición 1999

[U-EXPR] Extreme Programming
Disponible en: http://www.extremeprogramming.org
Ultima actualización: Febrero, 2006

[U-FPE] Fases de la Programación Extrema
Disponible en: http://programacionextrema.tripod.com/fases.htm

[U-AMXP]Agile Modeling and eXtreme Programming (XP)
Disponible en:
http://www.agilemodeling.com/essays/agileModelingXP.htm#PotentialFit
Ultima actualización: Marzo, 2007

[U-AM] Agile Modeling
Disponible en: http://www.ambysoft.com/books/agileModeling.html
Ultima actualización: Marzo, 2007

[U-SBG] Swing (biblioteca gráfica)
Disponible en: http://es.wikipedia.org/wiki/Swing_(biblioteca_gr%C3%A1fica)
Ultima actualización: Agosto 2006

[U-CGWS] Creating a GUI with JFC/Swing
Disponible en:
http://ji.ehu.es/LMAlonso/SW/java/Bib/tutorjava/html/ui/swingevents/eventsandcomponents.html
Ultima actualización: Noviembre 2002

[U-PWDN] Páginas Web Dinámicas

Dirección de cómputo para la administración académica

Coordinación de servicios de red

Universidad Nacional Autónoma de México

Disponible en: http://manuales.dgsca.unam.mx/webdina/concepto.htm

Ultima actualización: 2006

[U-PDD] Procesos de desarrollo: RUP, XP, FDD

Disponible en: http://www.javahispano.org/articles.article.action?id=76

Ultima actualización: Febrero 2003

[U-DHTML] DHTML

Disponible en: http://www.hipertexto.info/documentos/dhtml.htm

Ultima actualización: Julio 2007

[U-WikiXSL] XSL

Disponible en: http://es.wikipedia.org/wiki/XSL

Ultima actualización: Mayo 2007

[U-GPWXX] Generación de páginas Web usando XSLT y XML

Disponible en: http://geneura.ugr.es/~jmerelo/XSLT/

XSLT Tutorial

Disponible en: http://www.w3schools.com/xsl/default.asp

Ultima actualización: 2007

XSLT & XPath Tutorial

Disponible en: http://www.topxml.com/xsl/tutorials/intro/default.asp

[U-CCEJ] Convenciones de Codificación en Java

Disponible en: http://www.programacion.net/tutorial/convenciones/

Ultima actualización: 2006