



UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

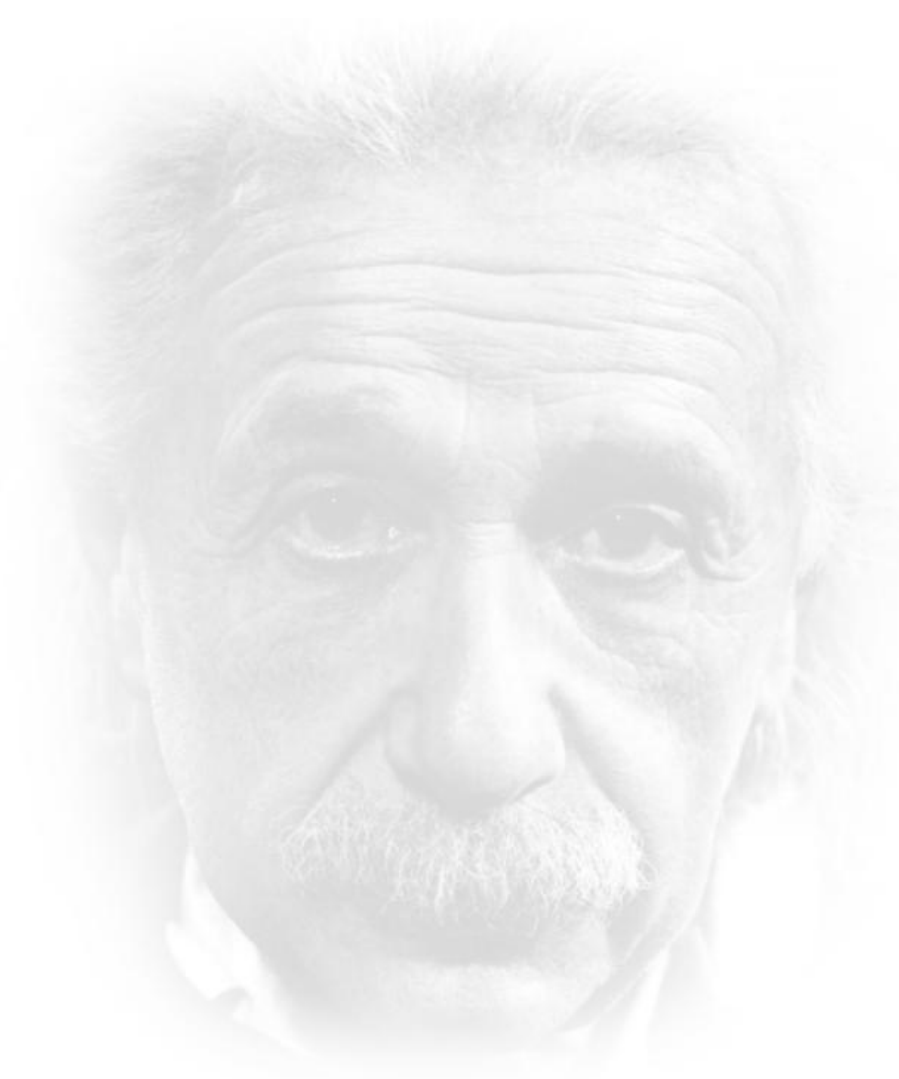
Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor: Roberley Cuadra Frías

Tutor: Ing. Julio Cesar Prieto Álvarez

La Habana, Cuba Junio 2012



“Cualquier tonto inteligente puede hacer cosas más grandes, más complejas, y más violentas. Se requiere un toque de genio y un poco de coraje para moverse en la dirección opuesta.”

Albert Einstein.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Firma del Autor

Firma del Tutor

Agradecimientos

Quiero agradecer principalmente a mi familia y a mi novia, por estar ahí cuando las necesito, por apoyarme siempre en mis decisiones, todos y cada uno son muy importantes para mí. También quiero agradecer a todos mis buenos amigos (estudiantes y profesores) que siempre me guiaron por buenos caminos y me dieron sus mejores consejos.

RESUMEN

Existen varias librerías en Java que brindan componentes para la creación de interfaces de usuario, entre las más conocidas se encuentran Swing y AWT. A pesar de que estas librerías son reconocidas por sus utilidades carecen de componentes, o los que contienen carecen de funcionalidades, que facilitan el desarrollo de las interfaces de usuario en los centros de desarrollo.

Para dar solución al problema planteado anteriormente se realizó este trabajo de diploma, cuyo objetivo principal fue desarrollar una librería de componentes de interfaz de usuario que facilitara el desarrollo de la capa de presentación en las aplicaciones de escritorio que la usen.

Esta librería de componente de interfaz gráfica de usuario (GUI por sus siglas en inglés: Graphic User Interface) está desarrollada sobre el lenguaje de programación Java 6.0, haciéndose uso de las librerías de componentes Swing y AWT, usando como guía la metodología de desarrollo de software XP y como entorno de desarrollo NetBeans 7.0.1. Finalmente la librería propuesta fue sometida a distintos tipos de pruebas obteniéndose resultados satisfactorios.

ÍNDICE

| | |
|---|----|
| INTRODUCCIÓN | 1 |
| 1 CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA | 5 |
| 1.1 INTRODUCCIÓN | 5 |
| 1.2 ESTADO DEL ARTE..... | 5 |
| 1.3 METODOLOGÍAS DE DESARROLLO | 7 |
| 1.3.1 <i>Proceso Unificado de Desarrollo</i> | 7 |
| 1.3.2 <i>Programación Extrema</i> | 8 |
| 1.3.3 <i>Scrum</i> | 9 |
| 1.4 LENGUAJE DE MODELADO | 10 |
| 1.4.1 <i>Lenguaje de Modelado Unificado (UML 2.0)</i> | 10 |
| 1.5 HERRAMIENTAS CASE..... | 11 |
| 1.5.1 <i>Rational Rose</i> | 12 |
| 1.5.2 <i>Visual Paradigm para UML</i> | 12 |
| 1.6 ESTILOS ARQUITECTÓNICOS | 13 |
| 1.6.1 <i>Arquitecturas Basadas en Componentes</i> | 14 |
| 1.6.2 <i>Arquitecturas en Capas</i> | 15 |
| 1.7 PATRONES..... | 15 |
| 1.7.1 <i>Patrones Arquitectónicos</i> | 16 |
| 1.7.2 <i>Patrones de Diseño</i> | 17 |
| 1.8 LENGUAJE DE PROGRAMACIÓN JAVA | 20 |
| 1.9 MARCOS DE TRABAJO..... | 21 |
| 1.9.1 <i>AWT</i> | 21 |
| 1.9.2 <i>SWING</i> | 22 |
| 1.9.3 <i>SWINGX</i> | 23 |
| 1.9.4 <i>SWT</i> | 23 |
| 1.9.5 <i>JUnit</i> | 24 |
| 1.10 ENTORNO DE DESARROLLO INTEGRADO | 25 |

| | | |
|--------|--|----|
| 1.10.1 | <i>NetBeans</i> | 25 |
| 1.10.2 | <i>Eclipse</i> | 25 |
| 1.11 | INTERFAZ DE PROGRAMACIÓN DE APLICACIONES | 26 |
| 1.11.1 | <i>API Reflexión</i> | 26 |
| 1.12 | CONCLUSIONES | 27 |
| 2 | CAPÍTULO 2: DISEÑO DE LA LIBRERÍA DE COMPONENTES | 28 |
| 2.1 | INTRODUCCIÓN | 28 |
| 2.2 | ARQUITECTURA DE LA LIBRERÍA | 28 |
| 2.3 | PATRONES | 28 |
| 2.3.1 | <i>Bajo Acoplamiento</i> | 28 |
| 2.3.2 | <i>Alta Cohesión</i> | 29 |
| 2.3.3 | <i>Iterador</i> | 29 |
| 2.3.4 | <i>Modelo Vista Controlador</i> | 29 |
| 2.4 | API REFLEXIÓN | 29 |
| 2.5 | COMPONENTES DE INTERFAZ DE USUARIO | 30 |
| 2.5.1 | <i>JBanner</i> | 30 |
| 2.5.2 | <i>JBarraTitulo</i> | 30 |
| 2.5.3 | <i>JBarraProgreso</i> | 31 |
| 2.5.4 | <i>JCedula</i> | 32 |
| 2.5.5 | <i>JComboBoxObjeto</i> | 32 |
| 2.5.6 | <i>JFormateador</i> | 33 |
| 2.5.7 | <i>JIP</i> | 33 |
| 2.5.8 | <i>JLabelToolTip</i> | 34 |
| 2.5.9 | <i>JMenu</i> | 34 |
| 2.5.10 | <i>JNumeroTramite</i> | 39 |
| 2.5.11 | <i>JPanelDegradado</i> | 40 |
| 2.5.12 | <i>JTextFieldInteligente</i> | 40 |
| 2.5.13 | <i>JPasswordFieldInteligente</i> | 41 |
| 2.5.14 | <i>JTextPaneAvanzado</i> | 41 |
| 2.5.15 | <i>JTablaObjeto</i> | 42 |
| 2.5.16 | <i>JListaObjeto</i> | 43 |

| | |
|---|----|
| 2.6 DISEÑO DE LA LIBRERÍA..... | 44 |
| 2.6.1 <i>Diagrama de Clases</i> | 44 |
| 2.6.2 <i>Diagrama de Componentes</i> | 48 |
| 2.7 ESTÁNDARES DE CODIFICACIÓN | 49 |
| 2.7.1 <i>Ficheros Fuente Java</i> | 50 |
| 2.7.2 <i>Declaraciones de Clases e Interfaces</i> | 50 |
| 2.7.3 <i>Ruptura de Líneas</i> | 50 |
| 2.7.4 <i>Comentarios</i> | 50 |
| 2.7.5 <i>Declaraciones</i> | 51 |
| 2.7.6 <i>Declaraciones de Clases e Interfaces</i> | 51 |
| 2.7.7 <i>Sentencias</i> | 51 |
| 2.7.8 <i>Espacios en Blanco</i> | 51 |
| 2.7.9 <i>Convenciones de Nombres</i> | 52 |
| 2.8 CONCLUSIONES | 52 |
| 3 CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA | 54 |
| 3.1 INTRODUCCIÓN | 54 |
| 3.2 PRUEBAS UNITARIAS | 54 |
| 3.2.1 <i>Prueba Unitaria JBanner</i> | 55 |
| 3.3 RESUMEN DE LAS PRUEBAS UNITARIAS | 56 |
| 3.4 PRUEBAS DE ACEPTACIÓN | 61 |
| 3.4.1 <i>Prueba de Aceptación JComboBoxObjeto</i> | 62 |
| 3.5 RESUMEN DE LAS PRUEBAS DE ACEPTACIÓN..... | 63 |
| 3.6 UTILIZACIÓN DE LOS COMPONENTES | 66 |
| 3.7 CONCLUSIONES | 67 |
| CONCLUSIONES..... | 68 |
| RECOMENDACIONES..... | 69 |
| BIBLIOGRAFÍA | 70 |

INTRODUCCIÓN

En la Universidad de las Ciencias Informáticas se han creado un conjunto de centros de desarrollo, con el objetivo de crear soluciones informáticas, en su mayoría vinculados a la informática jurídica, la gestión gubernamental, la salud y educación, que en muchos casos van más allá del contexto nacional.

En los centros de desarrollo se han creado ya una gran cantidad de aplicaciones de escritorio sobre la plataforma Java Standard Edition. Esta plataforma es una de las más robustas para el desarrollo de aplicaciones de escritorio que cumplan con las expectativas de la universidad. Estas expectativas se basan en las políticas de utilización de software libre que se están aplicando a lo largo de todo el país.

Una de las actividades más importantes dentro del proceso de desarrollo de una solución informática de escritorio es la creación de las interfaces de usuario. Los componentes contenidos en las interfaces de usuario son los que permiten el intercambio de información entre el sistema y los usuarios, por lo que estos deben ser lo más precisos posible, en aras de contribuir a la usabilidad de la solución informática.

Existen varias librerías que proporcionan componentes de interfaces de usuarios y que pueden ser integrados a la solución, de acuerdo a las arquitecturas definidas por las soluciones informáticas desarrolladas en la Universidad de las Ciencias Informáticas, como AWT, Swing y SwingX. Swing incluye widgets¹ para interfaz gráfica de usuario tales como cajas de texto, botones, desplegados y tablas. SwingX es una mejora de Swing que adiciona nuevas funcionalidades a los componentes existentes, y además incluye otros nuevos.

A pesar de que estas librerías son reconocidas por su utilidad, presentan inconvenientes que provocan que la creación de las interfaces gráficas de usuario en los centros de desarrollo se vuelva un trabajo engorroso. De las limitaciones que presentan las librerías de componentes existentes se pueden mencionar que algunos componentes como tablas y listas, no admiten una instancia completa de clases propias de las aplicaciones para mostrar su información, en su lugar es necesario crear otras funcionalidades que separen la información requerida de las instancias para poderla mostrar en estos

¹Un **widget** es un elemento gráfico con el que el usuario puede interactuar. Existen diversos tipos: botones, áreas de texto, etiquetas, etc.

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

componentes. También carecen de algunos componentes de interfaz gráfica de usuario que son muy utilizados en las aplicaciones que se desarrollan en la Universidad de las Ciencias Informáticas, de estos se pueden mencionar los componentes para la gestión de direcciones IP, números de trámites², así como un menú vertical que liste las funcionalidades de las aplicaciones. A todo esto mencionado se le puede añadir que los componentes que contienen no realizan funciones como validaciones avanzadas de texto mediante expresiones regulares.

Partiendo de la problemática planteada anteriormente se llega al planteamiento del siguiente **problema a resolver**: ¿Cómo facilitar el desarrollo de la capa de presentación de aplicaciones de software basadas en Swing Application Framework y AWT Framework?

Basado en lo anterior se ha definido como **objeto de estudio** de este trabajo el Desarrollo de software para aplicaciones de escritorio.

Enmarcado en el **campo de acción**: Desarrollo de componentes de interfaz gráfica para aplicaciones de software de escritorio.

Para dar solución a este problema planteado se ha propuesto como **objetivo general**: Desarrollar una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework que facilite el desarrollo de la capa de presentación de aplicaciones de software basada en estos.

Para dar solución a los elementos planteados anteriormente se debe dar cumplimiento a los siguientes **objetivos específicos**:

1. Elaborar el marco teórico de la investigación.
2. Desarrollar la librería de componentes como propuesta de solución.
3. Validar la solución propuesta.

Para este trabajo se plantea la siguiente **idea a defender**: Si se desarrolla una librería de componentes basada en Swing Application Framework y AWT Framework que elimine las limitaciones de los

² **Número de trámite**: es un número el cual sus primeros 3 dígitos significan el número de oficina en cuestión, los cuatro dígitos siguientes significan el año en que se genera el trámite, los 2 dígitos siguientes significan el trimestre y los dígitos restantes significan un valor consecutivo.

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

componentes contenidos en estos entonces se facilitará el desarrollo de la capa de presentación en las aplicaciones de software que la usen.

Para dar cumplimiento a los objetivos específicos planteados se proponen las siguientes tareas:

- Estudio del estado del arte de los marcos de trabajo para el desarrollo de interfaces gráficas de usuario para aplicaciones de escritorio en lenguaje Java.
- Identificación de los requisitos de software que deben reflejarse en cada componente a desarrollar.
- Identificación de los componentes Swing y AWT bases para la implementación de la librería.
- Definición de las metodologías, herramientas y patrones a utilizar para el desarrollo de la librería de componentes de interfaz de usuario propuesta.
- Diseño e implementación de los componentes de interfaz de usuario que formarán la librería propuesta.
- Diseño de casos de prueba de aceptación para cada componente a implementar.
- Diseño de casos de prueba unitaria para cada componente a implementar.
- Realización de las pruebas unitarias y de aceptación para cada componente a desarrollar en las soluciones de software SIGESAP v1.0 y DigiDAP v1.0 correspondientes al proyecto Registro de Antecedentes Penales.

En esta investigación se hace uso de varios **métodos científicos de la investigación**, los cuales se mencionan a continuación:

Métodos teóricos:

- *Inducción-Deducción:* Permitted llegar a conclusiones para la determinación del problema sobre las librerías existentes.
- *Analítico-Sintético:* Se realizó un estudio profundo de las principales herramientas y metodologías a utilizar en el desarrollo de la investigación basándose fundamentalmente en la revisión de informes, documentos y artículos.
- *Hipotético-Deductivo:* Desempeñó un rol importante en el proceso de verificación de la idea a defender, ya que a partir de esta se llegaron a nuevas conclusiones y predicciones que se fomentaron con la culminación de la investigación.

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

- *Histórico-Lógico*: Se realizó un estudio del estado del arte haciendo un análisis en las principales librerías de componentes de Java más utilizadas en la Universidad de las Ciencias Informáticas y a nivel mundial, teniendo en cuenta la trascendencia de estas librerías a lo largo de su existencia.

Métodos empíricos:

- *Observación*: Este método se utilizó para obtener una caracterización detallada de las soluciones existentes para librerías similares, con vista a valorar si a partir de ellas si se pueden eliminar los problemas existentes.

Con el desarrollo del presente trabajo se espera obtener una librería de componentes de interfaz de usuario para las soluciones informáticas que se desarrollan sobre la plataforma Java Standard Edition en la Universidad de las Ciencias Informáticas. Los componentes contenidos en esta permitirán gestionar la información intercambiada entre la aplicación informática y los usuarios manteniendo el enfoque orientado a objetos hasta las interfaces de usuarios. Con el uso de la librería se pretende dotar el desarrollo de interfaces de usuarios de una librería más usable y completa que las utilizadas hasta el momento, además de contribuir a la validación de la información gestionada por los usuarios.

La presente investigación estará dividida en tres capítulos. El **primer capítulo** titulado Fundamentación Teórica, expone todos los elementos teóricos que serán utilizados durante la investigación, así como el estado del arte sobre el campo de acción a diferentes niveles. El **segundo capítulo** titulado Diseño de la Librería de Componentes, describirá el diseño de la librería a partir de los artefactos propuestos por la metodología propuesta. El **tercer y último capítulo** se titula Validación de la Propuesta, el cual describe los principales algoritmos utilizados para la implementación de la librería así como las pruebas realizadas a cada uno de estos componentes.

1 CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

A lo largo del desarrollo de todo software es necesario realizar una serie de investigaciones que propicien los elementos teóricos para el desarrollo de la solución al problema planteado. En este capítulo se realiza un análisis crítico de los principales conceptos y tecnologías a utilizar. La metodología por la que se guiará el desarrollo, los principales marcos de trabajo que aportan elementos favorables a la solución, así como los principales elementos de otras tecnologías que influyen en el desempeño de la librería, son los temas fundamentales que se abordan.

1.2 Estado del Arte

Independientemente del auge existente en el desarrollo de aplicaciones web no se puede negar que el avance en el desarrollo de aplicaciones de escritorio es notable, fundamentalmente en el ámbito empresarial. Las interfaces de usuarios están conformadas por varios componentes que permiten el intercambio de información entre el usuario y la solución de software. Para el caso de Java, al usar herramientas libres, el diseño de dichas interfaces de usuario es un trabajo complejo para los desarrolladores, teniendo la necesidad de emplear mucho trabajo en el desarrollo de estas. Estos componentes deben ser de fácil comprensión para el desarrollador, facilitando la integración con otras capas de la solución de software, obligando a que su diseño sea lo más sencillo posible.

Para las aplicaciones de escritorio que se desarrollan en Java existen varias librerías que proporcionan componentes para interfaces de usuario. Entre las librerías más relevantes se encuentran Abstract Windows Toolkit (AWT, por sus siglas en inglés), Swing y SwingX.

La primera librería para el diseño de interfaces gráficas de usuario propuesta por Java fue AWT. Esta librería es un conjunto de clases que intentan ofrecer al desarrollador todo lo que necesita para crear una interfaz de usuario para cualquier applet³ o aplicación Java. La mayoría de los componentes AWT heredan de la clase `java.awt.Component`. La popularidad de AWT en su tiempo desbordó las expectativas de la propia empresa Sun Microsystems. Algunos de los problemas que presenta AWT es la falta de componentes avanzados como árboles y tablas, además consume excesivos recursos del sistema

³ Applet: Los applet son pequeñas aplicaciones en lenguaje de programación Java, que se ejecuta en los navegadores web.

operativo. Estos problemas dieron paso al surgimiento de Swing para la creación de interfaces de usuario en Java. (Sánchez Asenjó, 2004)

Swing es una librería de componentes desarrollada en java con el fin de crear interfaces de usuario. Algunas de las ventajas de Swing es que consume menos recursos, aporta más componentes que AWT y presenta una mejor apariencia a la solución de software. Esto no quiere decir que AWT sea remplazada, simplemente se reutilizan las funcionalidades de este en las nuevas capacidades de Swing. Sin embargo la interacción entre los componentes de esta librería y el resto de las capas de una aplicación se vuelve engorrosa debido a la complejidad de las interfaces que proporcionan.

Otra potente librería de componentes de Java para la creación de interfaces de usuario es SwingX, que se basa en una extensión del propio Swing. Esta nueva librería añade varias funciones no presentes en las otras anteriores librerías como por ejemplo funciones avanzadas de ordenación, selección, filtrado y resaltado a controles básicos de Swing como tablas, árboles y listas, añade también nuevas funcionalidades a otros componentes como cuadros de diálogo o paneles, e incluye otros componentes nuevos no presentes en Swing como por ejemplo JXTreeTable, JXLoginPanel o JXDatePicker para la presentación de tablas jerarquizadas, controles de autenticación o cuadros de selección de fecha. El principal problema que opaca el uso de la librería SwingX es que presenta poca documentación, esto trae consigo que no muchos desarrolladores decidan utilizarla debido a que es difícil de usar sus nuevas funcionalidades y componentes sin una documentación previa. (Sgoliver, 2009)

En la implementación del proyecto de Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela, a la hora de crear interfaces de usuario fue necesario utilizar una librería de componentes que facilitaran la comodidad de su uso a los desarrolladores. Las librerías antes mencionadas, aparte del gran número de componentes que contienen, carecen de algunos necesarios como por ejemplo un componente para el control de las direcciones de IP, un menú vertical para mostrar las funcionalidades de la solución de software, una tabla que acepte instancias completas de objetos del negocio. Analizando las fortalezas y debilidades de las librerías estudiadas, se ha tomado la decisión de desarrollar una librería de componentes, basados en Swing y AWT, que permita eliminar los problemas existentes en el desarrollo de las interfaces de usuario para la plataforma Java Standard Edition.

1.3 Metodologías de Desarrollo

Las Metodologías de Desarrollo de Software (MDS) nacen ante la necesidad de utilizar una serie de procedimientos, técnicas y herramientas para desarrollar un producto de software. Son diversos los escenarios en el proceso de desarrollo de un software, lo que ha dado al traste el surgimiento de las MDS que se ven clasificadas en la mayoría de las bibliografías en metodologías ágiles y metodologías pesadas.

Las metodologías pesadas son reconocidas por documentar la definición de los procesos y tareas que se van a realizar, estas requieren de una extensa documentación para describirlo todo. Estos tipos de metodologías son usadas mayormente en proyectos de largo alcance y que requieran de una gran organización, además donde la interacción con los clientes sea escasa. En cambio, las metodologías ágiles se encargan de valorar al desarrollador y las iteraciones del equipo más que la documentación de los procesos utilizados. Se caracterizan por ser metodologías en donde el cliente está en constante colaboración con el equipo de desarrollo y se hace mucho más importante elaborar un producto que funcione correctamente que generar tantos artefactos, estas metodologías son utilizadas mayormente en los proyectos de corto y medio alcance.

Entre las metodologías de desarrollo de software que más se destacan se pueden mencionar el Proceso Unificado de Desarrollo, Programación Extrema y Scrum. (Carrillo Pérez, y otros, 2008)

1.3.1 Proceso Unificado de Desarrollo

Proceso Unificado de Desarrollo (RUP, por sus siglas en inglés) es una de las metodologías de desarrollo de software pesada. RUP está caracterizado por estar centrado en la arquitectura y guiado por casos de uso. Esta metodología está compuesta por cuatro fases durante el desarrollo que define su ciclo de vida, estas fases son las siguientes:

Inicio: Se describe el negocio y se delimita el proyecto describiendo su alcance con la identificación de los casos de uso del sistema.

Elaboración: Se define la arquitectura del sistema y se obtiene un diseño que responde a los casos de uso que la comprometen.

Construcción: Se concentra en la elaboración de un producto totalmente operativo que responde a los requerimientos del cliente.

Transición: Se pone en producción el producto obtenido. Como consecuencia de esto suelen surgir nuevos requisitos a ser analizados en fases posteriores del desarrollo. (Gómes Gallego, y otros, 2007)

Estas cuatro fases de RUP son desarrolladas mediante un ciclo de iteraciones cada una. Cada iteración tiene como objetivo principal la evaluación de las iteraciones anteriores. Entre las características más relevantes de RUP se pueden destacar, la facilidad de reutilización al identificar partes comunes, facilita además el control de proyectos complejos, se basa en las técnicas de modelado propuestas por UML (aunque pueden utilizarse otras) y persigue la calidad del producto y del proceso en general.

Algunas de las ventajas que trae la utilización de la metodología RUP, es que disminuye la redundancia y aumenta la productividad en los proyectos, elimina ambigüedades en la comunicación del equipo de trabajo, une al equipo de trabajo facilitando su operación y monitoreo y logra una gran calidad en los productos de trabajo. (Itera, 2012)

1.3.2 Programación Extrema.

Programación Extrema (XP por sus siglas en inglés) pertenece al grupo de las metodologías ágiles. XP promueve el trabajo en equipo, asegurándose del aprendizaje de los desarrolladores y creando un clima de trabajo agradable. Este tipo de metodología es adecuada para los proyectos donde son imprecisos los requisitos, además de tener un riesgo técnico excesivo. Algunas de las características que presenta XP son las siguientes:

- **Desarrollo iterativo e incremental:** Se logran pequeñas mejoras, consecutivamente.
- **Pruebas unitarias continuas:** Estas se hacen muy frecuentemente y automatizadas, incluyendo pruebas de regresión.
- **Corrección de todos los errores:** Esto se hace antes de añadir nuevas funcionalidades.
- **Simplicidad en el código:** Es la mejor forma de que las cosas funcionen correctamente garantizando así la opción de añadir alguna nueva funcionalidad si fuera necesario.
- **Refactorización del código:** Consiste en reescribir algunas partes del código donde su comprensión no sea clara para aumentar su legibilidad y mantenibilidad, claramente sin modificar su comportamiento.

El uso de la metodología XP tiene como beneficio que la calidad de los sistemas basados en esta metodología tienden a ser un poco mejor debido a la cantidad de pruebas que se realizan consecutivamente durante todo el proyecto. Se lo logra una satisfacción con el cliente debido a que este tiene el control sobre las prioridades. Por otro lado, el desarrollo de software con XP es más flexible, y

como el sistema comienza a crecer orgánicamente, es más sencillo remover funciones para cumplir con el tiempo de desarrollo sin poner en riesgo el resto del sistema. (Martínez, 2002)

1.3.3 Scrum

Es otra de las metodologías que pertenece a las metodologías de desarrollo ágiles y está basada en un proceso iterativo incremental. Scrum está concebido para proyectos en los cuales los cambios de requisitos sean bastante frecuentes. Algunas de las características que presenta es que el desarrollo del software se realiza mediante iteraciones llamadas Sprints con un período de duración de treinta días aproximadamente, donde cada resultado de estos Sprints es un incremento ejecutable que se le muestra al cliente. Otra característica que describe a Scrum es que propone realizar varias reuniones a lo largo del proyecto, destacándose entre estas, una reunión diaria del equipo de desarrollo para la coordinación e integración del equipo.

Con la utilización de la metodología Scrum se produce el software de forma consistente, sostenida y competitiva, el trabajo del equipo de desarrollo es realizado en iteraciones poco extensas, de alto enfoque y total transparencia y el software se obtiene con la mayor rapidez posible y a su vez este cumple con los requerimientos más importantes. Por otra parte el uso de Scrum no es recomendado para equipos de trabajo que sean muy grandes, en aplicaciones que sean muy complejas, ni con un personal de trabajo con poca experiencia. (Gastón, 2008)

Para el desarrollo de la solución se propone usar como metodología de desarrollo a XP teniendo en cuenta que permitirá desarrollar componentes bastantes eficientes debido a la cantidad de pruebas continuas que propone como metodología. Con el uso de esta metodología se logrará enriquecer la librería gradualmente, sin poner en riesgo la utilización de esta en proyectos de software, además facilitará la retroalimentación en cuanto a la calidad de los mismos.

No se tuvieron en cuenta otras metodologías como RUP porque estas pudieran retrasar el desarrollo debido a que requieren mucha documentación que no representaría nada para los proyectos que la utilicen. Por otro lado los componentes de la librería deberán ser desarrollados, probados e incluidos a la misma de manera individual, por lo que el ciclo de desarrollo de estos es individual para cada uno de los componentes y la estructura de los mismos no es de gran complejidad.

1.4 Lenguaje de Modelado

El lenguaje de modelado es un conjunto de símbolos estandarizados, para modelar un diseño de software orientado a objetos. Este lenguaje en algunas organizaciones se usa extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. Para muchos usar este lenguaje de modelado es mucho más sencillo que la auténtica programación, esto se debe a que existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo.

1.4.1 Lenguaje de Modelado Unificado (UML 2.0)

El Lenguaje de Modelado Unificado (UML por sus siglas en inglés: Unified Modeling Language) es un lenguaje usado para definir, representar y documentar los diferentes aspectos relativos a un sistema de software en desarrollo, así como para modelado de negocios y almacenamiento de datos. Cada proyecto UML 2.0 es dividido en un número de diagramas que representan las diferentes vistas del proyecto y juntos representan la arquitectura del mismo, UML 2.0 brinda la posibilidad de describir un sistema en distintos niveles de abstracción, reduciendo la complejidad sin perder información, para que los desarrolladores y usuarios comprendan claramente las características de la aplicación.

UML 2.0 aporta varias vistas de un sistema los cuales ayudan a comprenderlo. Así, UML 2.0 recomienda la utilización de nueve diagramas para representar las distintas vistas de un sistema, estos diagramas son los siguientes:

Diagrama de Casos de Uso: es un diagrama sencillo, que tiene como objetivo dar una visión global de la aplicación a los usuarios y desarrolladores, y a la vez que puedan entenderlo de una forma rápida y gráfica.

Diagrama de Secuencia: es un diagrama para modelar la interacción entre los objetos, así como el paso de mensajes.

Diagrama de Colaboración: al igual que el Diagrama de Secuencia, muestra la interacción entre los objetos, resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.

Diagrama de Estados: este diagrama se utiliza para modelar los cambios de estado de los objetos y muestra los eventos, estados, transiciones y actividades de los diferentes objetos.

Diagrama de Actividades: este diagrama, simplifica el diagrama de Estados, modelando el comportamiento mediante flujos de actividades y muestra el flujo entre los objetos. También modela el funcionamiento del sistema y el flujo de control entre objetos.

Diagrama de Clases: describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

Diagrama de Objetos: este diagrama es un caso especial de un diagrama de Clases en el que se muestra una serie de objetos (instancias de las clases) y sus relaciones. Pero a diferencia de los diagramas demás diagramas, estos diagramas se enfocan en la perspectiva de casos reales o prototipos.

Diagrama de Componentes: este diagrama muestra las dependencias entre un conjunto de componentes, así como su organización.

Diagrama de Despliegue o Implementación: representa aspectos relativos a la implementación incluyendo la estructura del código fuente y otras características propias del tiempo de ejecución.

UML 2.0 es llamado hoy en día como el lenguaje estándar en el análisis y diseño de aplicaciones, el uso de este modelo para la creación de software brinda beneficios con respecto a: código reutilizable, descubrimiento de fallas, diseño y documentación, son fáciles las modificaciones, hay más comunicación entre los desarrolladores y se ahorra tiempo en el desarrollo del software. (Torazana, y otros, 2005)

1.5 Herramientas CASE

Las Herramientas de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés: Computer Aided Software Engineering) se pueden definir como un conjunto de ayudas y programas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de una aplicación de software. Estas herramientas permiten estandarizar y generar la documentación, facilitan la realización de prototipos y desarrollo conjunto de aplicaciones, simplifican el mantenimiento de los programas y aumentan la portabilidad de los programas. Las herramientas CASE no se pueden clasificar en una sola clase, estas son clasificadas de acuerdo a las plataformas que soportan, las fases del ciclo de vida del desarrollo de sistemas que cubren, la arquitectura de aplicaciones que producen o su funcionalidad. (Econ. Félix Murillo Alfaro, 1999)

1.5.1 Rational Rose

Rational Rose es una herramienta CASE que cubre todo el ciclo de vida de un proyecto, concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. Esta herramienta brinda la posibilidad de modelar y visualizar procesos de negocio, además permite destacar oportunidades para aumentar la eficiencia. Rational Rose permite el trabajo en equipo dando la posibilidad de que haya varias personas a la vez trabajando, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado y que tenga un control exclusivo sobre la propagación de los cambios en este espacio de trabajo. Otra de las utilidades que brinda el Rational Rose es la Ingeniería Inversa que permite, mediante el código de un programa, se obtenga información sobre su diseño. (Navarrete, 2006)

Rational Rose a pesar de ser una buena herramienta presenta inconvenientes, necesita de mucha memoria para poder ser manejado de forma rápida y eficiente. Si no se dispone de un buen rendimiento, presenta dificultades a la hora de editar los diagramas y trabajar con ellos.

1.5.2 Visual Paradigm para UML

“La mayoría de las herramientas CASE tienen funcionalidades comunes, generalmente denominadas como funcionalidades básicas para este tipo de herramientas. Sin embargo la generación de código java, así como la ingeniería inversa desde código fuente escrito en Java es más clara desde Visual Paradigm.” (Sierra, 2007)

El Visual Paradigm para UML 3.4 es una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Esta herramienta de modelado UML brinda la posibilidad de una rápida construcción de aplicaciones con una buena calidad. Visual Paradigm soporta gran cantidad de idiomas tanto en la generación de código como en la Ingeniería Inversa, algunos de estos idiomas a los cuales tiene la capacidad de soporte son Java, PHP, XML y Python. (López González, y otros, 2009)

Visual Paradigm 3.4 permite la conexión con otras herramientas en sus archivos de proyecto, también permite la importación y exportación de ficheros XML. Todo esto con el fin de maximizar la

interoperabilidad⁴ de los productos con otras aplicaciones; facilitando que los usuarios y proveedores de tecnología pueden integrar Visual Paradigm 3.4 en cada uno de sus modelos para utilizarlos en sus soluciones con un mínimo esfuerzo. Otra característica importante de esta herramienta consiste en que como es desarrollada en Java tiene la ventaja de poder ser ejecutado en varias plataformas como por ejemplo: Windows y Linux. Además tiene compatibilidad con varios IDEs⁵ de desarrollo como: Eclipse, NetBeans y Jbuilder.

Visual Paradigm para UML 3.4 es la herramienta CASE que se utilizó en la librería de componentes, para generar los diagrama de clases de cada uno de los componente, también se utilizó para generar el diagrama de componentes. Esta herramienta fue seleccionada debido a que era la herramienta de modelado definida por el proyecto donde fue creada la librería de componentes, se tuvo en cuenta además que la UCI tiene licencia académica de esta herramienta, es por eso que es conocida y utilizada por muchos proyectos.

1.6 Estilos Arquitectónicos

Los estilos arquitectónicos son una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles. Algunas de las ventajas de los estilos arquitectónicos son las siguientes: Los estilos arquitectónicos promueven reutilización de diseño (las soluciones de rutina con propiedades bien entendidas se pueden aplicar otra vez a nuevos problemas con alguna confianza). El uso de estilos arquitectónicos puede conducir a una significativa reutilización de código, además el uso de estilos estandarizados sustenta la interoperabilidad. Es usualmente posible, e incluso deseable, proporcionar visualizaciones específicas de un estilo. Esto proporciona representaciones gráficas y textuales que coinciden con intuiciones específicas de dominio sobre cómo deben representarse los diseños en él. (Reynoso , y otros, 2004)

⁴**Interoperabilidad:** se define como *la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.*

⁵ **IDE:** Entorno de Desarrollo Integrado IDE (por sus siglas en inglés: Integrated Development Environment).

1.6.1 Arquitecturas Basadas en Componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación a objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado. El estilo de arquitectura basado en componentes se caracteriza por ser un estilo de diseño para aplicaciones compuestas por componentes individuales. Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas y porque define una aproximación de diseño que usa componentes, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

En la Arquitecturas Basadas en Componentes, un componente es una pieza de código pre-elaborado que encapsula alguna funcionalidad expuesta a través de interfaces. Los componentes son los ingredientes de las aplicaciones, que se integran para llevar a cabo una tarea. A la hora de diseñar un componente hay que tener en cuenta sus principios básicos: estos tienen que ser reusables, ya que son diseñados para ser utilizados por diferentes aplicaciones en escenarios diferentes, además también algunos componentes pueden ser diseñados para tareas específicas. Los componentes deben ser extensibles debido a que estos pueden ser heredados desde otros componentes existentes para crear nuevos comportamientos. Otra de las especificaciones que debe presentar un componente es que debe ser independiente, debido a que estos pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas. (Casal Terreros, 2005)

La Arquitectura Basada en Componentes posibilita alcanzar un mayor nivel de reutilización de software. Permite que las pruebas sean ejecutadas de manera individual a cada uno de los componentes antes de probar el sistema integrado. Cuando existe un bajo acoplamiento entre componentes, el desarrollador es libre de actualizar o agregar componentes según sea necesario, sin afectar otras partes del sistema. Dado que un componente puede ser construido y luego mejorado continuamente, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo. (Casal Terreros, 2005)

1.6.2 Arquitecturas en Capas

La Arquitectura en Capas se define como una organización jerárquica tal, que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la capa inmediatamente inferior. Las capas en la práctica suelen ser entidades complejas, compuestas de varios paquetes.

El estilo puede incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma; esto se debe a que si esta exigencia se relaja, el estilo deja de ser correcto; también se pierde la posibilidad de reemplazar completamente una capa sin afectar a las restantes, disminuye la flexibilidad del conjunto y se complica su mantenimiento.

El uso del estilo en capas brinda varias ventajas. Primeramente, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los desarrolladores la partición de un problema complejo en una secuencia de pasos incrementales. Este estilo admite muy naturalmente optimizaciones y refinamientos y proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. (Reynoso , y otros, 2004)

La librería deberá estar compuesta por un conjunto de componentes de interfaz de usuarios de forma tal que se puedan mantener individualmente sin afectar el comportamiento de los demás por lo que la estructura de los mismos será individual, con las interfaces definidas para la interacción con otros componentes, lo que lleva a tomar la decisión de desarrollar la librería a través de una arquitectura basada en componentes.

1.7 Patrones

Dentro del marco de la ingeniería de software se puede definir un patrón de diseño como una solución reutilizable a un problema que ocurre frecuentemente dentro de un contexto en el diseño de software. Un patrón de diseño no es un diseño final que se puede incluir directamente en el código sino que se trata de un modelo o descripción para la forma de resolver un problema que puede ser utilizado en otras situaciones distintas. (Campo, 2009)

1.7.1 Patrones Arquitectónicos

1.7.1.1 Modelo Vista Controlador

La primera implementación del patrón Modelo Vista Controlador (MVC, por sus siglas en inglés) fue por los años ochenta en el lenguaje muy popular SmallTalk. Este patrón ha sido uno de los patrones más influyentes e importante en la historia de la programación, tanto ha sido así que hoy en día sigue siendo un patrón muy utilizado, tanto en aplicaciones de escritorios como en aplicaciones web. El patrón Modelo Vista Controlador pertenece a un conjunto de patrones agrupados en el estilo arquitectónico de presentación separada. El objetivo principal de este patrón es separar el código responsable de la representación de los datos, del código encargado de almacenarlos. Para lograr esto el patrón divide la capa de presentación en tres tipos de objetos básicos, estos son: modelos, vistas y controladores. La utilidad del patrón reside en que describe los flujos de comunicación entre estos tres tipos de objetos, como muestra esta imagen:

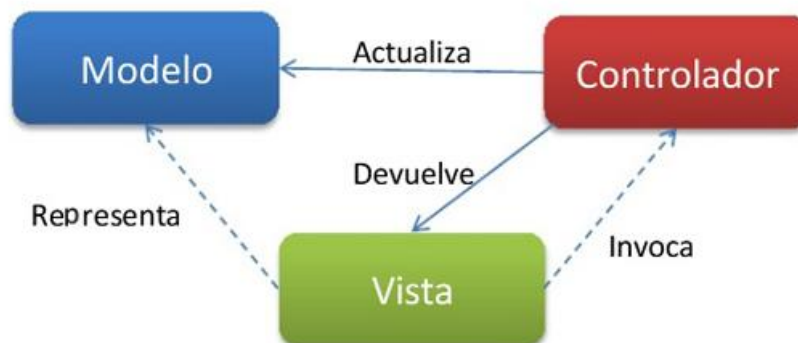


Fig.1 Patrón Modelo Vista Controlador

En un escenario clásico la vista representa los datos del modelo e invoca acciones de un controlador en respuesta a las acciones del usuario. El controlador recoge las acciones del usuario, interactúa con el modelo y devuelve una determinada vista en respuesta. En la mayor parte de las implementaciones de

MVC, los tres componentes pueden relacionarse directamente el uno con el otro, pero en algunas implementaciones el controlador es responsable de determinar que vista mostrar. En fin, este patrón será capaz de conseguir un bajo acoplamiento en sus aplicaciones, esto lo logra desacoplando los modelos de las vistas, reduce la complejidad en el diseño arquitectónico e incrementa la flexibilidad y mantenimiento del código. (de la Torre LLorente, y otros, 2010)

1.7.2 Patrones de Diseño

1.7.2.1 Patrón Fachada

El patrón Fachada proporciona una interfaz unificada de alto nivel para un componente, que oculta las interfaces de bajo nivel de las clases que lo implementan. Con esto se consiguen dos objetivos fundamentales: hacer el componente más fácil de usar y desacoplar a los clientes de las clases del subsistema. Este patrón pertenece a la categoría de patrones con propósito estructural y alcance de objeto, esto se debe a que propone una manera de componer a sus participantes, en este caso objetos, mediante relaciones determinadas dinámicamente en tiempo de ejecución. (Ramos Fernández, 2006)

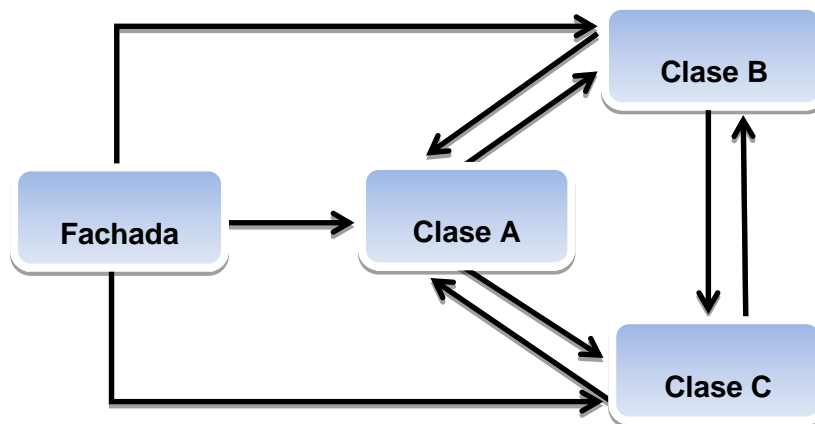


Fig.2 Patrón Fachada

El patrón Fachada será aplicado cuando se quiera estructurar varios componentes en capas, ya que las fachadas serían el punto de entrada a cada nivel, o cuando se necesite proporcionar una interfaz simple para un subsistema complejo. Otro escenario propenso para la aplicación de este patrón surge de la necesidad de desacoplar un sistema de sus clientes y de otros subsistemas, haciéndolo más independiente, reutilizable y portable.

Este patrón de diseño trae consigo varias ventajas mirándolo de diferentes puntos de vista. Mirando desde el punto de vista del subsistema, el uso de fachadas ayuda a controlar o eliminar las dependencias complejas o circulares entre objetos, ayuda a organizar un sistema en capas y permite hacer cambios en los componentes de un subsistema sin afectar a los clientes. Por otra parte desde el punto de vista de los clientes, la fachada los aísla de los componentes del subsistema minimizando el número de objetos con los que tienen que tratar para obtener un servicio. De esta manera los clientes son mucho más fáciles de escribir pues el subsistema es más sencillo de utilizar. Mirándolo desde un punto de vista en general, el uso de fachadas facilita enormemente el desarrollo independiente de clientes y subsistemas y tiene consecuencias muy importantes sobre la reusabilidad de los subsistemas, objetivo principal perseguido por todos los patrones de diseño.

Pero no todo es ventajas, también se pueden apreciar algunas desventajas y es que introduce un nivel de indirección⁶ adicional que podría afectar al rendimiento de la aplicación además oculta parte de la funcionalidad de un subsistema. Esto es cierto para clientes que se decanten por la facilidad de uso de la fachada, pero nada les impide mirar más allá de la misma para aprovechar toda la generalidad del subsistema. (Ramos Fernández, 2006)

1.7.2.2 Patrones GRASP

“Los Patrones Generales de Software para Asignar Responsabilidades (GRASP, por sus siglas en inglés) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.” (Saavedra Gutierrez , 2007)

1.7.3.1 Bajo Acoplamiento

El acoplamiento se puede medir por la cantidad de relaciones que presenta una clase hacia otras, mientras menos relaciones tenga una clase respecto a las otras menor será el acoplamiento. El alto acoplamiento trae como consecuencia que los cambios que ocurran en cualquiera de las clases con las que se relaciona una clase determinada, puedan proporcionar errores a dicha clase, además cuando las

⁶ La **indirección** es una técnica de programación. El concepto se basa en hacer referencia indirecta a los datos usando las direcciones de memoria que los contienen o mediante punteros que señalan hacia esos datos o a las direcciones que los contienen.

clases tienen muchas relaciones con otras, son más difíciles de reutilizar, debido a que se requiere de la presencia de las otras clases de las que dependen.

Para evitar problemas de acoplamiento se aconseja utilizar el patrón Bajo Acoplamiento. Este patrón propone que el desarrollador asigne responsabilidades a clases de tal modo que su colocación no incremente el acoplamiento, evitando que se produzcan los resultados negativos propios de un alto acoplamiento como por ejemplo cuando se realizan cambios en una clase de la cual dependen otras, muchas veces estos cambios afectan a las otras clases. Dicho patrón soporta el diseño de clases más independientes, que disminuyen el impacto de los cambios, y también más reutilizables, que amplíen la oportunidad de una mayor productividad.

“El uso de este patrón bien aplicado permite que las aplicaciones sean más mantenibles, extensibles y reutilizables, ya que tienen perfectamente definidas las responsabilidades y eso permite que puedan ser reutilizadas en cualquier otro contexto, siempre que la funcionalidad que se les demande sea la misma”.
(Jiménez Rodá, 2008)

1.7.3.2 Patrón Alta Cohesión

Para realizar un buen diseño en la programación orientada a objeto se aconseja que cada elemento debe realizar una labor única dentro del sistema, no desempeñada por otros de los elementos. La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Para que se pueda decir que una clase tenga una alta cohesión, esta tiene que tener responsabilidades estrechamente relacionadas al objeto que representa o describe. Por otra parte una clase que tenga baja cohesión realiza un trabajo excesivo, estas clases mayormente son difíciles de comprender, reutilizar y son muy delicadas porque las afectan constantemente los cambios.

Para evitar este problema se aconseja utilizar el patrón Alta Cohesión. EL patrón consiste en valorar cada clase y darle a cada una un alto grado de cohesión, con una importante funcionalidad principal y con poco trabajo que hacer. En caso de que la tarea sea muy grande, se debe relacionar con otros objetos para compartir el esfuerzo. La utilización de este patrón mejora en gran medida la claridad y la facilidad con que se puede entender el diseño, se hace más fácil el mantenimiento y se mejoran las funcionalidades, incluyendo que en ocasiones se propicia un bajo acoplamiento.

1.7.2.3 Patrones GoF

Los patrones GoF (por sus siglas en inglés: Gang of Four) son un conjunto de patrones recopilados y documentados por cuatro expertos diseñadores de software orientado a objetos: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Los patrones de GoF se clasifican en 3 categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Creacionales: Tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.

Estructurales: Describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionales pueden ser incluso objetos simples u objetos compuestos.

Comportamiento: Ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos. (Angelfire, 2011)

1.7.2.3.1 Patrón Iterador

En ocasiones es necesario recorrer una colección de datos sin importar la estructura en donde estos están almacenados. Este problema se puede solucionar haciendo uso del patrón Iterador. Este patrón pertenece a los patrones de diseño GoF, pertenece a la categoría de comportamiento. Estos patrones son utilizados para organizar, manejar y combinar comportamientos. Los patrones Iteradores permiten recorrer colecciones de objetos sin importar la estructura en donde estos están almacenados; estos patrones se encuentran mayormente en clases genéricas. La implementación de este patrón incrementa en gran medida la flexibilidad del código, dado que para permitir nuevas formas de recorrer una estructura basta con modificar el iterador en uso, cambiarlo por otro o definir uno nuevo.

1.8 Lenguaje de Programación Java

Java 6.0 es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. Algunas de las características más relevantes de Java 6.0 es que es un lenguaje orientado a objeto, es independiente de la plataforma, esto significa que se puede ejecutar en cualquier sistema operativo que contenga la Máquina Virtual de Java (JVM por sus siglas en inglés). En Java 6.0 mediante la utilización del recolector de basura se resuelve el problema de las fugas de

memoria⁷, por ejemplo si el recolector de basura en tiempo de ejecución ve que un objeto pierde todas las referencias, lo elimina liberando así la memoria ocupada. Java toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. (Zukowski, 2003)

1.9 Marcos de Trabajo

Los marcos de trabajo son diseñados principalmente con el objetivo de facilitar el desarrollo de software, dándoles la posibilidad a los desarrolladores de pasar más tiempo identificando requerimientos de software, que no tratando con los engorrosos detalles de bajo nivel de proveer un sistema funcional. Los marcos de trabajo suelen incluir soporte de programas, bibliotecas, software para desarrollar y unir diferentes componentes de un proyecto de desarrollo de software. En pocas palabras no es más que una base de programación que atiende a sus descendientes (manejado de una forma estructural y/o en cascada), posibilitando cualquier respuesta ante las necesidades de sus miembros, o n secciones de una aplicación, satisfaciendo así las necesidades más comunes del desarrollador.

1.9.1 AWT

El Kit de Herramientas de Ventana Abstracta (AWT por sus siglas en inglés: Abstract Window Toolkit) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. Se trata de una biblioteca portátil para aplicaciones GUI independientes y/o applets. AWT proporciona la conexión entre la aplicación y la interfaz gráfica de usuario nativa, también proporciona un alto nivel de abstracción para el programa Java, ya que esconde los detalles fundamentales de la interfaz gráfica de usuario de los programas que se ejecutan en él. El Kit de Herramientas de Ventana Abstracta presenta un amplio conjunto de componentes de la interfaz de usuario y un robusto modelo de gestión de eventos. También incluye los gráficos y herramientas de imágenes, incluyendo su forma, color, y las clases de fuentes. Además incluye administradores de diseño, para los diseños de las ventanas flexibles, que no dependen de un tamaño de ventana en particular o resolución de la pantalla y clases de transferencia de datos, para cortar y pegar a través del portapapeles. El Kit de Herramientas de Ventana Abstracta carece de algunos componentes avanzados como árboles y tablas,

⁷ Una **fuga de memoria** (más conocido por el término inglés *memory leak*) es un error de software que ocurre cuando un bloque de memoria reservada no es liberada en un programa de computación. Comúnmente ocurre porque se pierden todas las referencias a esa área de memoria antes de haberse liberado.

además consume excesivos recursos del sistema operativo y presenta también algunos problemas de compatibilidad en algunos sistemas operativos. Estos factores influenciaron que los componentes de AWT fueran ampliamente superados por los nuevos componentes de Swing. (García de Jalón, y otros, 2000)

1.9.2 SWING

Swing es una librería de componentes gráficos que ha aparecido en la versión 1.2 de Java. Swing fue creado para desarrollar interfaces gráficas en Java con independencia de la plataforma. Está integrado por un amplio conjunto de componentes de interfaces de usuario que funcionan en el mayor número posible de plataformas. Cada uno de los componentes de esta librería puede presentar diversos aspectos y comportamientos en función de una biblioteca de clases. En la versión 1.0 de Swing, que corresponde a la distribuida en la versión 1.2 de la API de Java se incluyen tres bibliotecas de aspecto y comportamiento para Swing, estas son *metal.jar*, *motif.jar* y *windows.jar* con características muy similares a las interfaces de Microsoft Windows. (Manzanedo del Campo, 1999)

Es importante destacar que Swing sigue un simple modelo de programación por hilos y que presenta tres características que lo representan, estas son las siguientes: Swing es independiente de plataforma, esto es muy importante ya que no se rige por ningún sistema operativo, puede correr en cualquier plataforma sin problema alguno. Su segunda característica es la extensibilidad dado a que posee una arquitectura altamente particionada, esto es factible para los usuarios ya que estos pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto y se puede extender clases existentes proporcionando alternativas de implementación para elementos esenciales. La tercera característica que presenta Swing es que es personalizable dado el modelo de representación programático del marco de trabajo de Swing, el control permite representar diferentes estilos de apariencia llamados "Look and Feel"⁸. Además, los usuarios pueden proveer su propia implementación de apariencia, que permitirá cambios uniformes en la apariencia existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.

⁸ **Look and Feel:** Se utiliza en relación a una interfaz gráfica de usuario e incluye aspectos de su diseño, incluyendo elementos como los colores, formas, diseño, y tipos de letra, así como el comportamiento de los elementos dinámicos como botones, cajas y los menús.

1.9.3 SWINGX

SwingX es un marco de trabajo de componentes gráficos para Java. Este marco de trabajo es una extensión de Swing, donde se mejoran muchos de los controles existentes en Swing, y se incluyen algunos controles nuevos. De las nuevas funciones avanzadas que integra SwingX se pueden mencionar las de ordenación, filtrado, selección y resaltado a controles básicos de Swing como a tablas, árboles y listas, añade además nuevas funcionalidades a otros controles como cuadros de diálogo o paneles, e incluye algunos controles nuevos no presentes en Swing como JXTreeTable, JXLoginPanel o JXDatePicker para la presentación de tablas jerarquizadas, controles de autenticación o cuadros de selección de fecha. (Sgoliver, 2009)

El objetivo del proyecto es experimentar funcionalidades GUI nuevas o mejoradas que son requeridas por las aplicaciones. Actúa como un banco de pruebas para ideas relacionadas con las aplicaciones de escritorio.

1.9.4 SWT

SWT (por sus siglas en inglés: *Standard Widget Toolkit*) es una librería de componentes equivalente a la librería Swing en la cual, se aprovechan los componentes nativos del sistema operativo sobre el que se ejecuta. Aprovechar los componentes nativos, permite que la ejecución de interfaces de usuario sea mucho más rápida y fluida que si se utilizase Swing y, además, siempre dispone del Look and Feel del sistema, sin necesidad de emularlo. Esta librería fue desarrollada para el IDE Eclipse y no es compatible con ningún otro como por ejemplo NetBeans. Además SWT es nativa, lo quiere decir, que es necesario disponer de una librería SWT específica para cada sistema operativo. Aunque existen versiones de SWT para los sistemas operativos más habituales, entre ellos Windows y Linux. (Retamar, 2009)

Para seleccionar las librerías de componentes base se realizó un estudio riguroso sobre estas librerías antes mencionadas, comparando las ventajas y desventajas de cada una. De estas se decidieron utilizar dos de ellas, AWT y Swing. AWT porque brinda numerosas clases necesarias para la creación de componentes, entre ellas la clase Color, Font, MouseEvent entre otras. Swing se decidió utilizar por la parte de los componentes, debido a que contiene componentes que no están presente en AWT como es el caso de las tablas y los árboles.

Las librerías SWT y SwingX no se tuvieron en cuenta a la hora de desarrollar la librería porque SWT es una librería que fue creada para el IDE Eclipse y no tiene compatibilidad con otros IDE como NetBeans. SwinX por su parte carece de una buena documentación que facilite el uso de sus componentes por los desarrolladores. A continuación se muestra una tabla de comparación entre las librerías analizadas.

| Librerías | Compatibilidad con NetBeans | Buena Documentación |
|-----------|-----------------------------|---------------------|
| Swing | SI | SI |
| AWT | SI | SI |
| SWT | NO | SI |
| SwingX | SI | NO |

Tabla 1: Tabla de comparación entre las librerías

1.9.5 JUnit

La fase de prueba es una de las fases más importantes en del desarrollo de software, esto se debe a que en esta es donde se verifica que el proyecto cumpla con los requerimientos que fueron especificados por el cliente. Existen varios tipos de pruebas, sin embargo, algunas de las más básicas que existen son las pruebas unitarias, que prueban los distintos módulos que conforman el proyecto y que son el inicio para la correcta realización de otros tipos de pruebas. Existen varias herramientas para realizar estas pruebas unitarias, permitiendo que el esfuerzo y el trabajo en la fase de pruebas se reduzcan, permitiendo que el desarrollador se centre en el análisis de los resultados. Para realizar estas pruebas unitarias en el lenguaje de programación Java se utiliza JUnit.

JUnit es la herramienta de prueba más utilizada para la realización y ejecución de pruebas unitarias sobre el código Java, esta herramienta viene incluida muchos IDEs, entre ellos se encuentran NetBeans y Eclipse. Esta herramienta forma parte del conjunto conocido como XUnit, que es un conjunto de frameworks para realizar pruebas unitarias en varios lenguajes de programación, como por ejemplo PUnit para PHP o NUnit para .Net. JUnit contiene varias funcionalidades para realizar pruebas unitarias sobre el

código Java, estas pruebas se realizan de manera controlada sobre los distintos métodos que conforman una clase, haciendo una comparación entre el valor esperado y el valor que realmente retorna el método. JUnit permite generar reportes de las pruebas realizadas. Estos reportes se obtienen en formato HTML donde se especifican las distintas pruebas realizadas a cada clase y sus resultados, estos están organizados según el paquete al que pertenezcan. (Hernández, 2009)

1.10 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado, llamado también IDE (por sus siglas en inglés: Integrated Development Environment). Es una aplicación que está compuesta por un conjunto de herramientas que les son útiles a un desarrollador. Los IDEs pueden ser exclusivamente definidos para un solo lenguaje de programación o bien, pueden utilizarse para varios lenguajes como por ejemplo: Java, Python, C++, Visual Basic, Delphi, C# y otros. Los IDEs suelen estar compuestos de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario. (Doar, 2005)

1.10.1 NetBeans

NetBeans 7.01 es un proyecto de código abierto con una gran cantidad de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun Microsystems fundó el proyecto de código abierto NetBeans en junio 2000, pero actualmente el patrocinador de este proyecto es Oracle al comprar la Sun Microsystems. NetBeans 7.0.1 aunque está pensado para tecnologías Java tiene soporte para XML, lenguajes HTML, C y C++, JSP, JDBC y otros. Cuenta con un editor de código sensible al contenido con autocompletamiento de código, autotabulación, coloreado de etiquetas y uso de abreviaturas para varios lenguajes de programación. Incluye control de versiones lo que facilita el trabajo en equipo. Con NetBeans se pueden crear además interfaces de usuarios debido a que nos brinda la posibilidad de utilizar componentes gráficos para aplicaciones de escritorio de gran calidad. Brinda la posibilidad de utilizar proyectos creados por otros IDEs como es el caso de Eclipse. (NetBeans, 2010)

1.10.2 Eclipse

Eclipse es un IDE al que se le pueden integrar herramientas de desarrollo para varios lenguajes de programación mediante el uso de los plugins⁹ adecuados. Estos plugins permiten además introducir otras herramientas que son útiles durante el proceso de desarrollo como son las herramientas para modelado

⁹ **Plugins:** Son extensiones que añaden funciones adicionales a Eclipse

UML, editores visuales de interfaces entre otros. Eclipse presenta compilación en tiempo real, resaltado de sintaxis, editor de texto, pruebas unitarias con JUnit, control de versiones, asistentes para la creación de proyectos, refactorización de clases, entre otros más. Una de las ventajas del uso de Eclipse es que requiere menos recursos que otros IDEs de desarrollo, como por ejemplo NetBeans.

Para la implementación de la librería de componentes se determinó utilizar NetBeans 7.01 como IDE, teniendo en cuenta que este incluye las librerías bases (Swing y AWT) en el mismo paquete de instalación, evitando el procedimiento de integración de este en otros IDEs como Eclipse. El resaltado de líneas, el completamiento de código y la adaptabilidad hacia los estándares de codificación definidos es más usable desde NetBeans que desde Eclipse.

1.11 Interfaz de Programación de Aplicaciones

Cuando se habla de Interfaz de Programación de Aplicaciones (API por sus siglas en inglés: Application Programming Interface) se refiere a un conjunto de funciones y procedimientos (o métodos llamados así en la programación orientada a objetos) que ofrece una biblioteca (también llamadas comúnmente librerías) para ser utilizado por otro software como una capa de abstracción. Son usadas mayormente en las bibliotecas. (Mastermagazine, 2004)

1.11.1 API Reflexión

El API Reflexión le brinda la posibilidad a un programa de que en tiempo de ejecución pueda acceder a toda la información de sí mismo, como también a la del API del lenguaje y de las librerías o bibliotecas a las que tenga acceso. No solamente permite el acceso a la información sino que también permite manipularla. Admite además invocar dinámicamente métodos y constructores de otras clases u objetos. Con todo esto se pueden hacer cosas que van desde saltar los modificadores de acceso de una definición de clase, hasta hacer que dos objetos intercambien información de atributos privados sin que medien métodos *setter*. (Green, 2004)

Algunas de las funcionalidades que nos brinda el API Reflexión son las siguientes:

- Determina la clase de un objeto en tiempo de ejecución.
- Obtiene información acerca de modificadores de una clase, campos, métodos, constructores y superclases.

- Obtiene y establece el valor del campo de un objeto, incluso si el nombre del campo es desconocido para el programa hasta el tiempo de ejecución.
- Crea una nueva matriz, cuyo tamaño y tipo de componente no se conocen hasta el tiempo de ejecución, y modifica los componentes de la matriz. (Green, 2004)

Las utilidades que brinda este API son bastante avanzadas, por lo que se recomienda que deben ser usadas por desarrolladores que tengan un buen conocimiento de los fundamentos del lenguaje. A esto se le puede sumar que trabajar con objetos dinámicamente en este código, provoca que en muchas ocasiones el control de errores no funcione correctamente en tiempo de compilación y que también se complique en cierta medida en el tiempo de ejecución. Teniendo esto bien controlado se puede decir que Reflexión es una API muy poderosa y que con ella las aplicaciones pueden realizar operaciones que de otro modo sería imposible realizar. (Oracle, 2010)

1.12 Conclusiones

En este capítulo se analizaron cada una de las metodologías, frameworks y herramientas, que tienen mayor uso en el desarrollo de software, y fueron seleccionadas para el desarrollo de la librería según sus características. De las metodologías se seleccionó XP, por ser una metodología ágil recomendada para proyectos de corto alcance. Con respecto a la arquitectura se seleccionó la arquitectura basada en componentes debido a que el desarrollo basado en esta permitirá reutilizar componentes desarrollados previamente sin tener en cuenta la composición de los mismos, sumándole un incremento considerable en la mantenibilidad del código. Para implementar la librería de componentes se utilizó como IDE de desarrollo NetBeans 7.0.1 por ser el IDE oficial de Java y por contener las librerías bases (Swing y AWT).

2 CAPÍTULO 2: DISEÑO DE LA LIBRERÍA DE COMPONENTES

2.1 Introducción

En el presente capítulo se describe cada uno de los componentes que la librería contendrá y la interacción entre ellos a través de diagramas UML. Por cada componente se podrá encontrar el diagrama de clases que lo compone, así como la descripción detallada de cada una de estas y los principales métodos. También se describirán los patrones usados por la librería y los estándares de codificación que se tuvieron en cuenta a la hora de desarrollar la librería. Todo esto permitirá a los desarrolladores tener un mejor entendimiento de cómo usar la librería y de las funcionalidades que esta brinda.

2.2 Arquitectura de la Librería

La utilización de una arquitectura basada en componentes en la librería permitió reutilizar componentes desarrollados previamente, entre los que se pueden mencionar algunos como JTable, JTextField y JPanel, sin tener en cuenta la composición de los mismos, también se incrementó considerablemente la mantenibilidad de la librería, debido a que la actualización de estos componentes será transparente al ser cada uno independiente de los demás.

2.3 Patrones

2.3.1 Bajo Acoplamiento

Durante del desarrollo de la librería de componentes se puso en práctica el patrón Bajo Acoplamiento, este fue utilizado para proporcionar un bajo acoplamiento entre las clases, tratando de que cada una de ellas tengan la menor cantidad de relaciones posibles con otras clases. Un ejemplo donde se puede observar este patrón dentro de la librería de componentes es en la clase JBanner, esta clase se pudiera relacionar con tres clases para su funcionamiento (JPanelDegradadoRadial, GradienteCircularPaint, GradienteCircularContexto), pero en cambio haciendo uso del patrón, de estas tres solo se relaciona con una, que es la clase JPanelDegradadoRadial. Aplicando el mismo concepto a las demás clases que también se relacionan, se obtuvieron las siguientes relaciones: la clase JPanelDegradadoRadial se relaciona únicamente con la clase GradienteCircularPaint, por su parte la clase GradienteCircularPaint se relaciona solamente con GradienteCircularContexto. Haciendo uso de este patrón en la librería no se afectan los componentes al realizarse cambios en otros, además son más fáciles de reutilizar y de mantener por separados.

2.3.2 Alta Cohesión

Durante el desarrollo de la librería de componentes se puso en práctica el patrón Alta Cohesión, con el objetivo de lograr una alta cohesión en la mayor cantidad de clases de la librería. Un ejemplo donde se puede observar el uso de este patrón es en el componente JMenu donde cada una de las clases que lo integran tienen una función específica con funciones relacionadas, esto es uno de los primeros principios del patrón Alta Cohesión. La clase JScrollPaneContenedor se encarga solamente de contener al componente JPanelContenedor y proporcionar las barras de desplazamiento al este sobrepasar sus dimensiones. El componente JPanelContenedor se encarga de organizar los componentes JPanelGrupo que contenga para que todos estén unos debajo del otro a una medida determinada. Y por último el componente JPanelGrupo el encargado de organizar los componentes JBotonLink que contenga, para que estos también estén ubicados unos debajo de otros a una determinada distancia.

2.3.3 Iterador

A lo largo del desarrollo de la librería de componentes se hizo necesario recorrer colecciones de objetos sin tener en cuenta la estructura que los almacenaba, para esto se utilizó el patrón Iterador. Este patrón fue utilizado en las clases genéricas JTablaObjeto, JListaObjeto y JComboBoxObjeto, para poder recorrer cualquier colección de objetos, ya sea una lista, una cola o una pila.

2.3.4 Modelo Vista Controlador

EL patrón Modelo Vista Controlador fue utilizado en la librería de componentes con el objetivo de separar el código encargado de la representación de los datos, del código encargado de almacenarlos. Un ejemplo del uso de este patrón se puede observar en la implementación del componente JListaObjeto donde el modelo lo representa la clase DefaultListModel que es una clase definida por Java. Esta clase es la encargada de almacenar los datos que van a ser mostrados en la lista y la clase encargada de representar estos datos, es la clase JListaObjeto que representaría la vista y el controlador, esta última función la realiza también esta clase debido a que captura los eventos de los usuarios y modifica tanto la vista como el modelo.

2.4 API Reflexión

El API Reflexión fue utilizado en la implementación de los componentes JListaObjeto, JTablaObjeto y JComboBoxObjeto de la librería. Estos son genéricos y admiten cualquier listado de instancia de clases propias de las aplicaciones; sin embargo los componentes no conocen cuáles son los atributos de las

instancias de las clases que van a ser mostrados en ellos. Haciendo uso del API Reflexión este problema se resuelve debido a que permite acceder a la información de una instancia de una clase en tiempo de ejecución, esto facilita que los desarrolladores puedan especificar en las propiedades de estos componentes cuales son los atributos de las instancias de las clases que van a ser mostrados.

2.5 Componentes de Interfaz de Usuario.

2.5.1 JBanner

Los banners son imágenes que se sitúan en la parte superior de las aplicaciones. En muchas aplicaciones de escritorio es necesario mostrar en algún momento un banner para la aplicación, ya sea para darle estética, propaganda o para mostrar alguna información. Este componente brinda la posibilidad de mostrar un banner para las aplicaciones de escritorio. La ventaja que brinda utilizar este componente es que permite que al redimensionar la aplicación no se distorsione la imagen del banner. Esto lo hace mediante una técnica que consiste en dividir la imagen del banner en dos partes y colocarla en ambos lados del componente dejando la parte central del componente vacío y con un color del fondo similar al color de fondo de la imagen del banner. Esta parte central del componente es la que se redimensiona con la aplicación brindando la posibilidad de que se ajuste este componente a la aplicación sin que la imagen del banner pierda calidad. A continuación se muestra una imagen del componente:



Fig.3 Componente JBanner

2.5.2 JBarraTitulo

Las barras de título forman parte de las ventanas de las aplicaciones y se ubican en la parte superior de estas. Estas barras proporcionan información acerca del programa que se esté ejecutando y también permiten opciones como cerrar, maximizar y minimizar las ventanas. En algunas ocasiones es necesario modificar la barra de título de la aplicación ya sea para cambiarle el color, la estética de la ventana, para mostrar alguna información distinta sobre ella o simplemente para darle otro enfoque a la aplicación, esto resulta imposible o simplemente muy engorroso debido a que las ventanas que usan las aplicaciones de escritorio son las ventanas del sistema operativo y se hace muy difícil cambiar las propiedades de estas. JBarraTitulo es una barra de título que permite la modificación de sus propiedades, como el color, el

tamaño, se le puede cambiar el icono y la fuente del texto que contenga. A continuación se muestra una imagen del componente:



Fig.4 Componente JBarraTitulo

2.5.3 JBarraProgreso

Una barra de progreso es un componente que es usado para indicar el progreso de una aplicación que se esté ejecutando. Swing en sus grupos de componentes contiene una barra de progreso que brinda varias funcionalidades que son usadas en las aplicaciones de escritorio. Esta barra de progreso al igual que todos los componentes de Swing son controlados por los Look and Feel, esto trae como consecuencia que al aplicarle estos controladores a una ventana, todos los componentes que estén contenidos en esta se comporten visualmente de la misma manera. Al ocurrir esto es imposible modificar los efectos visuales de estos componentes entre los que se encuentra las barras de progreso. JBarraProgreso es una barra de progreso similar a la que contiene la librería Swing, pero la principal característica que la diferencia, es que su comportamiento visual no depende de los Look and Feel. Esto tiene como ventaja que independientemente de los estilos visuales que se estén utilizando en una ventana, este componente puede ser modificado por los desarrolladores y será mostrado visualmente sin ser afectado por los controladores. A continuación se muestra una imagen del componente:

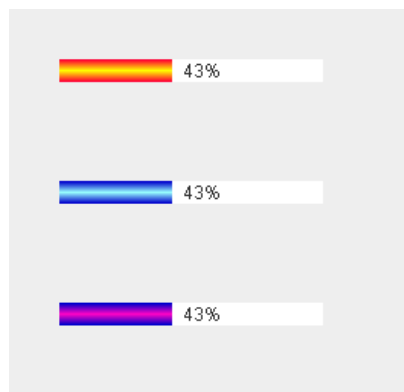


Fig.5 Componente JBarraProgreso

2.5.4 JCedula

JCedula no se encuentra en ninguna librería de componentes de interfaz gráfica de usuario. Este componente es creado con el objetivo de facilitar el trabajo con las cédulas de identidad¹⁰ de los ciudadanos de la República Bolivariana de Venezuela. JCedula está formado por dos partes, la primera parte (la parte izquierda) muestra una lista desplegable con los posibles estados que puede tener un ciudadano, estos estados pueden ser: venezolano, extranjero o indocumentado. Este listado que se muestra es configurable ya que se le puede pasar por parámetros al componente. La segunda parte del componente (la parte derecha) es la parte que permite la entrada del número de cédula, esta parte valida que solo se le puedan insertar números al componente. A continuación se muestra una imagen del componente:

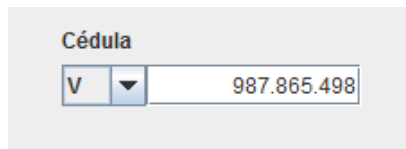


Fig.6 Componente JCedula

2.5.5 JComboBoxObjeto

Un Combo Box es un componente que permite la selección de datos mostrando una lista desplegable con los datos a seleccionar, también permite la entrada textual de información. Swing cuenta con un Combo Box dentro de sus componentes, este trae funcionalidades que facilitan su uso en las aplicaciones. Este Combo Box de Swing no admite la entrada de datos que no sean nativos del sistema como por ejemplo enteros, cadenas de texto, entre otros. JComboBoxObjeto es un Combo Box que admite la entrada de instancias de una clase de cualquier tipo y permite mostrar en su lista desplegable los atributos que se le indique por sus propiedades. Esto posibilita no perder tiempo en descomponer un objeto en atributos que queramos mostrar; a este componente solo se le pasa la lista de objetos y se le señala cual es el atributo de estos que debe ser mostrado. A continuación se muestra una imagen del componente:

¹⁰ **Cédula de Identidad** (C.I.): es un documento de identidad emitido por el SAIME para la identificación personal de los ciudadanos de Venezuela.

SAIME: Servicio Administrativo de Identificación, Migración y Extranjería.

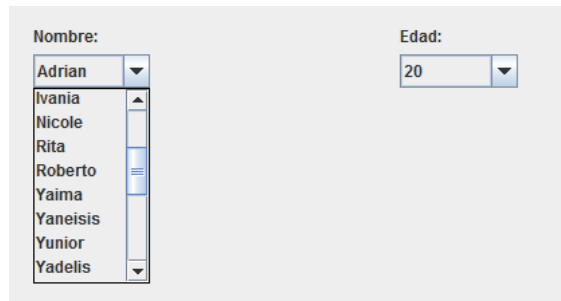


Fig.7 Componente JComboBoxObjeto

2.5.6 JFormateador

En ocasiones se necesita restringir la entrada de algunos datos o simplemente la forma de entrada de estos y realizar la validación de estas operaciones suele ser muy difícil. Las librerías existentes no cuentan con un validador de texto tan potente como JFormateador. Este componente como se dijo anteriormente permite la validación de cualquier texto que sea entrado por el usuario. Las validaciones que realiza dicho componente se efectúan mediante expresiones regulares que son insertadas en el componente en una de sus propiedades. El texto que no cumpla con las propiedades puede ser eliminado automáticamente o puede ser notificado según desee el desarrollador. A continuación se muestra una imagen del componente:

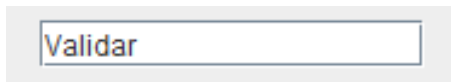


Fig.8 Componente JFormateador

2.5.7 JIP

Swing no cuenta con un componente para la gestión de direcciones IP, es por eso que surge JIP para manejar estas direcciones con mayor exactitud. Este nuevo componente no está presente en ninguna de las librerías estudiadas; trae implementado un gran número de restricciones que aseguran un correcto manejo de las direcciones IP. JIP está conformado por cuatro cuadros de texto que son los que permiten la entrada de estos números. Cada cuadro de texto permite la entrada solamente de números y a la vez valida que cada número esté en un rango entre 0 y 255 que son los rangos admisibles que tienen que cumplir cada octeto de las direcciones IP. Cada cuadro de texto está diseñado también para que no permitan campos de números vacíos, una vez que ocurra esto, ellos automáticamente en dicho lugar

asignan un valor de cero para que no ocurran errores con dicho número IP. A continuación se muestra una imagen del componente:

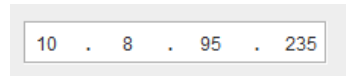


Fig.9 Componente JIP

2.5.8 JLabelToolTip

Cuando se está desarrollando un proyecto de largo alcance a la hora de desarrollar interfaces de usuario es necesario aprovechar la mayor cantidad de tiempo posible. JLabelToolTip es una etiqueta que tiene como característica que muestra como texto de ayuda, el texto que contenga, sin tener que configurarlo previamente por el desarrollador. Esto logra que en el desarrollo de una interfaz de usuario en la que hay varias etiquetas (esto ocurre generalmente) se ahorre el tiempo considerablemente, no teniéndole que insertar a cada etiqueta además de su texto normal, el texto de ayuda, debido a que el mismo componente realiza esta operación por sí solo. El componente también permite mostrar el texto de forma vertical. A continuación se muestra una imagen del componente:

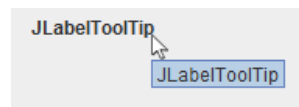


Fig.10 Componente JLabelToolTip

2.5.9 JMenu

Un menú vertical es un componente que permite mostrar en su interior una lista de acciones que pueden ser ejecutadas por un usuario. Los menús verticales se utilizan mayormente en aplicaciones donde estas contengan muchas interfaces, y sea necesario tener un controlador para poderse ir desplazando entre ellas. Swing en su lista de componentes no contiene un menú vertical para sus aplicaciones, esto dio motivo para crear al componente JMenu que es un menú vertical muy eficiente compuesto por un grupo de componentes por separado. Este menú cuenta con funcionalidades como la inserción de botones, permite agrupar estos botones en grupos según sus características (mayormente llamados módulos), permite nombrar estos grupos así como insertarles un icono que los represente, lo mismo puede hacerse para los botones. JMenu permite la modificación de los colores de todos los componentes que lo integran,

a estos, se les puede configurar un degradado que da una mejor apariencia a las aplicaciones que los usen. A continuación se muestra una imagen del componente:

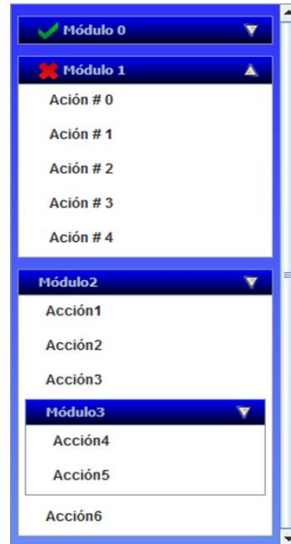


Fig.11 Componente JMenu

JMenu está compuesto por cinco componentes, que se muestran a continuación:

2.5.9.1 JPanelGrupo

JPanelGrupo es un agrupador de componentes similar a los de Swing. La función que realiza dentro del componente JMenu es ordenar todos los componentes que le sean insertados en su interior, esto con el objetivo de tenerlos organizados y para ahorrar espacio en el menú debido a que este tiene la capacidad de minimizarse, ocultando todos los componentes que contenga. A continuación se muestra una imagen del componente:

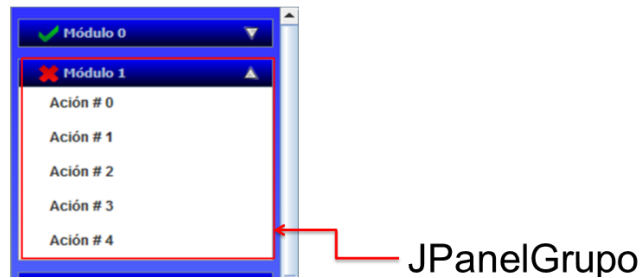


Fig.12 Componente JPanelGrupo

2.5.9.2 JBotonLink

Este componente es el que ejecuta la acción que se le programe. JBotonLink presenta las mismas características funcionales de un botón de la librería Swing, pero con respecto a la visualización este componente es transparente y solo muestra un texto para identificar la acción que este realiza. Este nuevo componente puede ir contenido dentro de un agrupador de acciones (JPanelGrupo), esto depende de la ubicación que quiera darle el desarrollador. A continuación se muestra una imagen del componente:

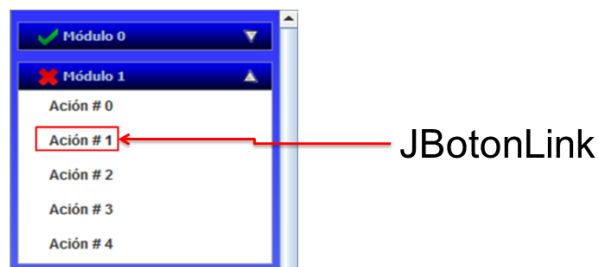


Fig.13 Componente JBotonLink

2.5.9.3 JBotonDegradado

Este componente se sitúa en la parte superior del JPanelGrupo y es el que controla los eventos de maximizar y minimizar que puede recibir el agrupador. Es usado también como barra de título del JPanelGupo, para dar información acerca de los componentes que contenga este. Puede mostrar iconos en ambas parte de sus extremos, en la parte izquierda muestra un icono normalmente igual que todos los botones y en la parte derecha muestra un icono que es el que indica si esta maximizado o minimizado el JPanelGrupo. Este botón puede tomar dos aspectos, el primero es el

aspecto normal que toman todos los botones de Swing y el segundo que es el que mejor apariencia le brinda, es formado por un degradado, al cual se le puede configurar los colores tanto los que presenta normalmente como los que debe mostrar al ser seleccionado. A continuación se muestra una imagen del componente:

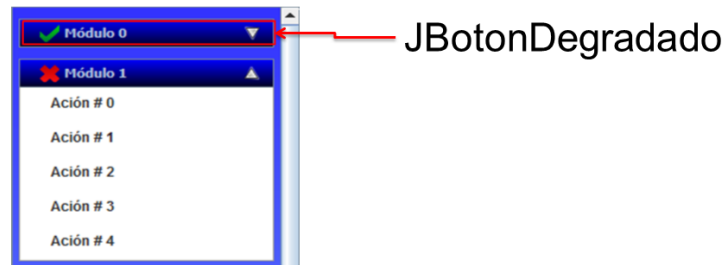


Fig.14 Componente JBotonDegradado

2.5.9.4 JPanelContenedor

JPanelContenedor es un contenedor de componentes y es el encargado de controlar los demás componentes que se muestran a través de él, ejemplo de estos son los JPanelGrupo y los JBotonLink. Este componente organiza los componentes que contenga, de tal manera que quedan situados unos encima de otros a una distancia que puede ser configurada por los desarrolladores. Al insertar o eliminar algún componente dentro de JPanelContenedor automáticamente se ajustan los restantes, logrando así que siempre esté organizado el menú. Este componente presenta un fondo degradado al cual se le puede configurar los colores según guste el desarrollador. A continuación se muestra una imagen del componente:

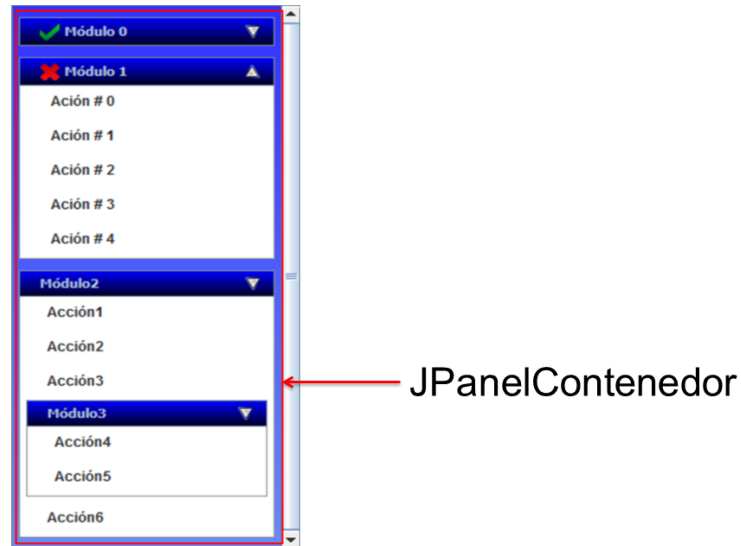


Fig.15 Componente JPanelContenedor

2.5.9.5 JScrollPaneContenedor

Cuando un JPanelContenedor contiene demasiados componentes dentro, ajusta su tamaño para no dejar ninguno sin visualizar y entonces sobrepasa sus dimensiones predefinidas. Un JScrollPaneContenedor contiene a un JPanelContenedor y su función consiste, en que al percibir que este componente que contiene sobrepasa el límite sus dimensiones, muestra automáticamente barras de desplazamiento, para que el usuario pueda desplazarse por el menú y visualizar todas las partes de este que no se muestran. A continuación se muestra una imagen del componente:

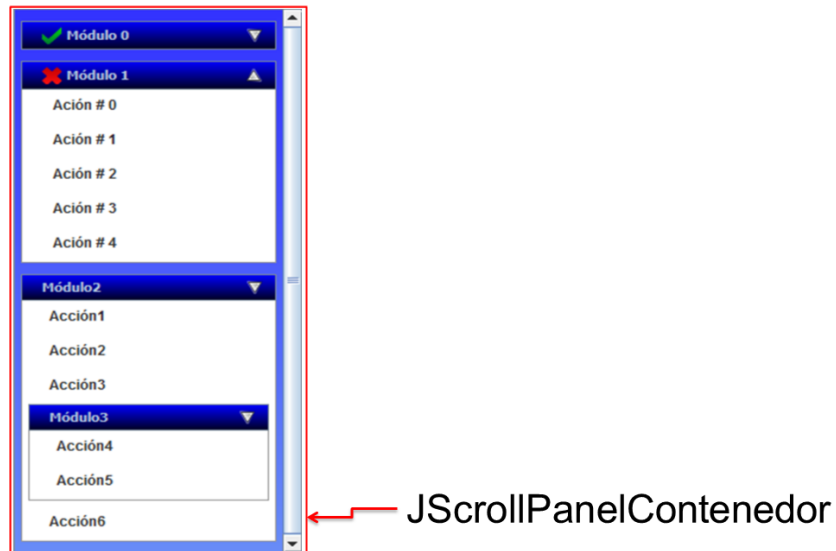


Fig.16 Componente JScrollPaneContenedor

2.5.10 JNumeroTramite

Este componente es usado mayormente en aplicaciones donde se requiera trabajar con trámites y estos se identifican con alguna secuencia de caracteres. JNumeroTramite es creado con el objetivo de facilitar el trabajo con los números de trámites, ya que valida que los formatos de estos números estén escritos correctamente para evitar posibles errores en las aplicaciones. Este componente está compuesto por cuatro campos de texto que son los que rigen la entrada correcta de los números de trámite. Estos campos de texto no permiten la entrada de otros caracteres que no sean números, evitando así la posibilidad de que puedan entrar caracteres especiales, signos o letras. A cada campo se le puede restringir la cantidad de dígitos que este puede admitir, posibilitando esto que dicho componente pueda ser usado por un desarrollador para otros fines. A continuación se muestra una imagen del componente:



Fig.17 Componente JNumeroTramite

2.5.11 JPanelDegradado

Un JPanel es un contenedor de otros componentes de interfaz gráfica de usuario que viene integrado en la librería de swing. Como todos los contenedores, utiliza un Controlador de Distribución que es el encargado de posicionar y dimensionar sus componentes. JPanelDegradado es un contenedor de componentes también, en específico este componente tiene como característica principal que su fondo es un degradado al cual se le puede modificar los colores, el ángulo del degradado, el radio y la dimensión. Este componente es base de todos los componentes de la librería que trabajan con degradado. A continuación se muestra una imagen del componente:

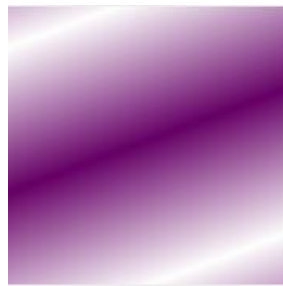


Fig.18 Componente JPanelDegradado

2.5.12 JTextFieldInteligente

Un JTextField es uno de los componentes de la librería de Swing que permiten la entrada de texto. Estos componentes son normalmente conocidos como campos de texto, sus controles pueden mostrar y editar sólo una línea de texto y están basados en acción como los botones. Los campos de texto son utilizados normalmente para obtener una pequeña cantidad de información textual insertada por un usuario, como por ejemplo el nombre, la edad y el correo. JTextFieldInteligente es un campo de texto que presenta características visuales más agradables a los usuarios. Este componente muestra un texto (insertado por defecto por un desarrollador) indicando la información que requiere, y cuando el usuario comienza a escribir, este texto se oculta automáticamente, y si el usuario borra lo escrito, aparecerá el texto por defecto nuevamente. Los JTextFieldInteligentes además de dar un toque de modernidad a las aplicaciones por sus efectos, ahorran espacio en las interfaces de usuario de las aplicaciones debido a que las etiquetas de texto que se utilizan para indicar el nombre de los campos de texto, no se utilizan porque estos componentes hacen ellos mismos dicha función. A continuación se muestra una imagen del componente:



Fig.19 Componente JTextFieldInteligente

2.5.13 JPasswordFieldInteligente

Un JPasswordField es uno de los componentes con que cuenta la librería de Swing. Este componente es usado para la entrada de contraseñas de los usuarios. JPasswordField es una subclase de JTextField, pero en vez de mostrar el carácter real tecleado por el usuario, muestra otro carácter que por lo general siempre es un asterisco; esto tiene como objetivo ocultar la identidad de la contraseña del usuario para que no pueda ser vista fácilmente por otra persona. JPasswordFieldInteligente es también usado para la entrada de contraseñas, pero con características visuales más agradables. Las características que presenta este componente son similares a las de JTextFieldInteligente. Las aplicaciones que usan estos componentes mejoran respecto a su apariencia y obtienen un toque de modernidad, brindando una mejor apariencia a los usuarios finales. A continuación se muestra una imagen del componente:

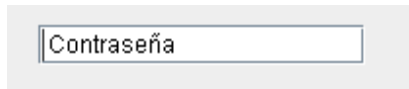
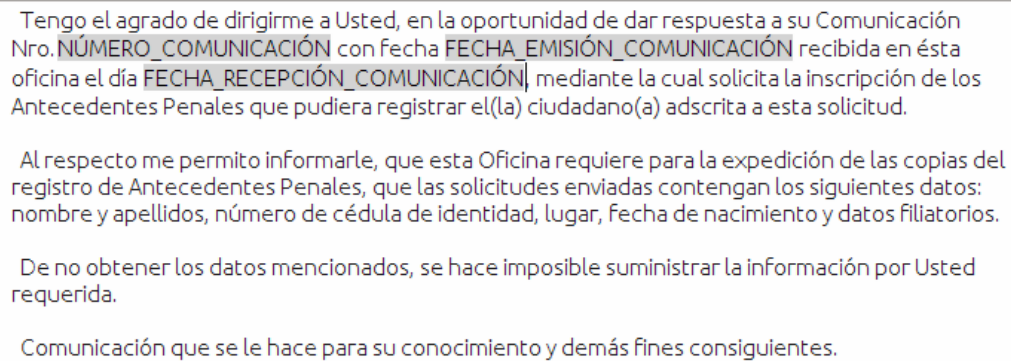


Fig.20 Componente JPasswordFieldInteligente

2.5.14 JTextPaneAvanzado

Un JTextPane es un componente de texto formateado, puede mostrar y editar texto usando más de una fuente. Estos componentes de texto formateado permiten embeber imágenes e incluso componentes. Se tendrá que realizar una programación más compleja para usar y configurar componentes de texto formateado, porque muchas de sus funcionalidades no están disponibles a través de la manipulación directa con el ratón y el teclado. JTextPaneAvanzado es un componente que extiende las funcionalidades provistas por el JTextPane que proporciona la librería Swing. Este permite editar texto insertando tokens¹¹ dentro del mismo de manera visual facilitando esta tarea a los usuarios, además permite el reemplazo de dichos tokens por valores en tiempo de ejecución, siendo apropiado para la creación de textos con elementos variables. A continuación se muestra una imagen del componente:

¹¹ **Token:** Se refiere a una palabra o un grupo de palabras que pueden ser reemplazadas en tiempo de ejecución.



Tengo el agrado de dirigirme a Usted, en la oportunidad de dar respuesta a su Comunicación Nro. `NÚMERO_COMUNICACIÓN` con fecha `FECHA_EMISIÓN_COMUNICACIÓN` recibida en ésta oficina el día `FECHA_RECEPCIÓN_COMUNICACIÓN` mediante la cual solicita la inscripción de los Antecedentes Penales que pudiera registrar el(la) ciudadano(a) adscrita a esta solicitud.

Al respecto me permito informarle, que esta Oficina requiere para la expedición de las copias del registro de Antecedentes Penales, que las solicitudes enviadas contengan los siguientes datos: nombre y apellidos, número de cédula de identidad, lugar, fecha de nacimiento y datos filiatorios.

De no obtener los datos mencionados, se hace imposible suministrar la información por Usted requerida.

Comunicación que se le hace para su conocimiento y demás fines consiguientes.

Fig.21 Componente JTextPaneAvanzado

2.5.15 JTablaObjeto

JTable es uno de los nuevos componentes que integró Swing que no estaba presente en AWT. Este componente es uno de los más complejos, su funcionalidad se basa en que puede mostrar tablas de datos, y opcionalmente permitir que el usuario los edite. Para mostrar una tabla de datos usando JTable de la librería de Swing hay que crear un arreglo bidimensional y llenarlo con los datos que se quieran mostrar de la lista de instancias de clases propias de las aplicaciones. Esta labor se hace muy engorrosa y se pierde mucho tiempo en realizarla. JTablaObjeto es usado también para mostrar tablas de datos a los usuarios, pero presenta muchas más funcionalidades que no contiene un JTable común de la librería Swing. Una de las funcionalidades más importante que permite este nuevo componente es la entrada de una lista de instancias de clases que pueden ser del negocio, y permite una configuración para que en cada una de sus columnas muestre los atributos de estas instancias (los que el desarrollador le indique). A continuación se muestra una imagen del componente:

| Nombre | Edad | Sexo | Nota |
|----------|------|------|------|
| Adrian | 20 | M | 4 |
| Amys | 25 | F | 3 |
| Ariel | 22 | M | 4 |
| Cesar | 26 | M | 3 |
| Emeterio | 21 | M | 5 |
| Ernesto | 26 | M | 3 |
| Ivania | 24 | F | 4 |
| Nicole | 23 | F | 5 |
| Rita | 24 | F | 5 |
| Roberto | 27 | M | 3 |
| Yaima | 25 | F | 4 |
| Yaneisis | 24 | F | 5 |
| Yunior | 29 | M | 3 |
| Yadellis | 23 | F | 3 |
| Barbara | 27 | F | 5 |
| Felipe | 25 | M | 3 |
| Jose | 21 | M | 4 |
| Emeterio | 21 | M | 5 |
| Ernesto | 26 | M | 3 |
| Ivania | 24 | F | 4 |
| Nicole | 23 | F | 5 |

Fig.22 Componente JTablaObjeto

2.5.16 JListaObjeto

Un JList es un componente presente en la librería de swing. Estos permiten mostrar un listado de información en forma de ítems a los usuarios los cuales pueden ser seleccionados. JListaObjeto es una lista que admite instancias de cualquier clase y en su listado de información muestra los atributos de esta instancia que le sean indicados. Cualquier cambio que se realice en este componente, como por ejemplo eliminar un objeto, trabajará directamente sobre la lista de objetos que posee. A continuación se muestra una imagen del componente:

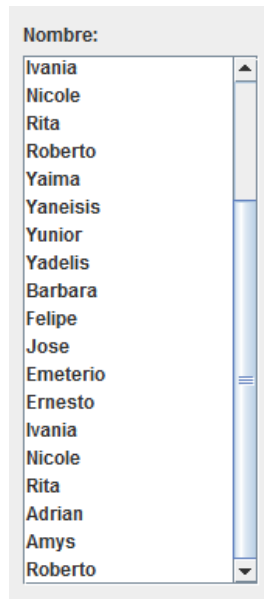


Fig.23 Componente JListaObjeto

2.6 Diseño de la Librería

La librería de componentes está compuesta por 19 paquetes; dentro de ellos hay 17 que contienen las clases que forman los componentes y 2 paquetes auxiliares, uno que contiene imágenes que son utilizadas en algunos componentes y el otro contiene clases que son útiles para la librería. La librería contiene un total de 20 componentes que les facilita a los desarrolladores el trabajo en los proyectos.

2.6.1 Diagrama de Clases

Los diagramas de clases son utilizados para describir la estructura de un sistema, mostrando de este sus clases, atributos y las relaciones que existan entre ellos. Los diagramas de clases están compuestos por dos elementos, las clases y las relaciones. Una clase es una unidad básica que encapsula toda la información de un Objeto y las relaciones son usadas para representar de una manera entendible, la relación que puede presentar una clase con otra o con varias.

Los diagramas de clases de cada uno de los componentes de la librería se encuentran en los anexos del presente trabajo, a continuación se muestra un ejemplo de uno de estos:

2.6.1.1 JFormateador

Es un componente que mediante expresiones regulares realiza validaciones al texto que le es entrado por el usuario. Este componente está compuesto por tres clases las cuales se describen a continuación.

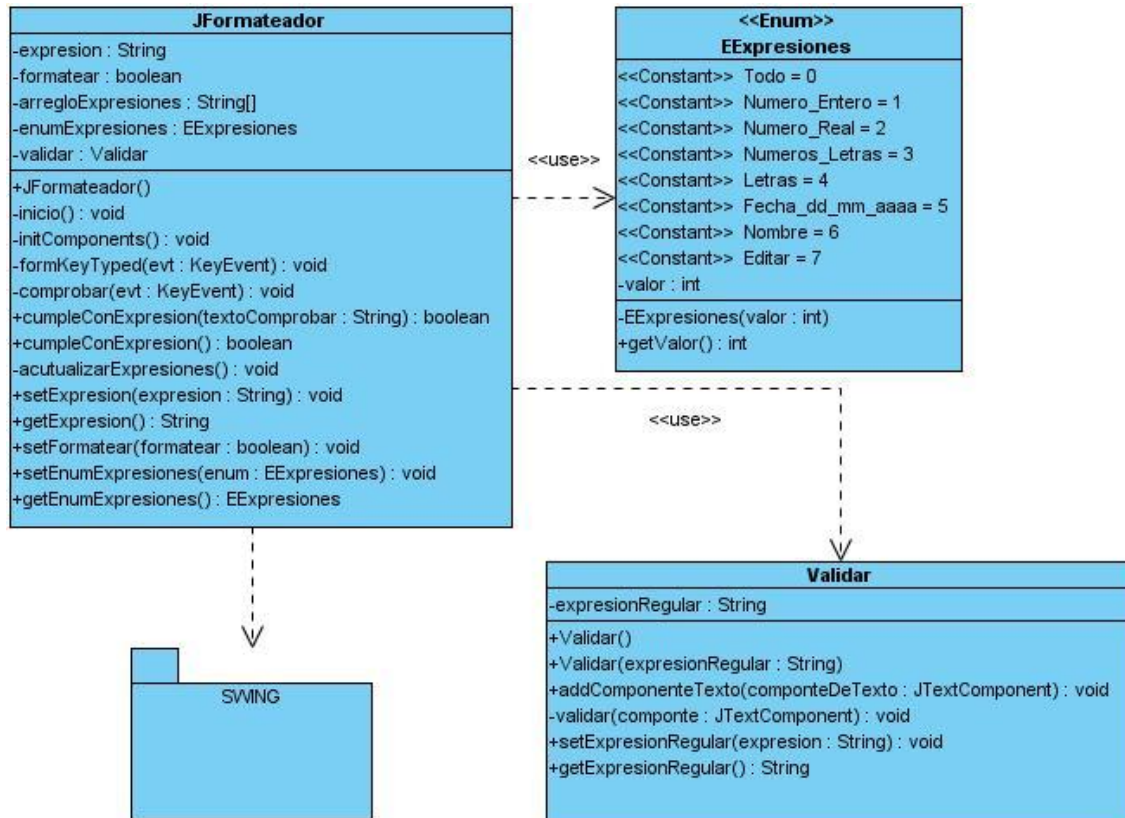


Fig.24 Diagrama de clases del componente JFormateador

A continuación se muestran las tablas de descripción de cada una de las clases que conforman a los componentes. Estas tablas son utilizadas para describir el objetivo principal de cada clase y para describir los métodos contenidos en estas.

Nombre de la clase: JFormateador

Descripción: Clase que representa la interfaz de las funcionalidades del componente hacia otros componentes u otras capas de las aplicaciones de escritorio que utilicen la librería. Esta clase hereda

| de JTextField y se apoya en las clases EExpresiones y Validar para realizar su funcionamiento. | |
|--|---|
| Método | Descripción |
| JFormateador() | Constructor. |
| comprobar(KeyEvent evt) | Valida que no entren caracteres al componente que no cumplan con la expresión regular definida. |
| cumpleConExpresion(String textoComprobar) | Devuelve verdadero si el texto entrado por parámetro cumple con la expresión regular definida en el componente. |
| cumpleConExpresion() | Devuelve verdadero si el texto que contiene el componente cumple con la expresión regular definida. |
| setExpresion(String expresion) | Define la expresión regular del componente. |
| getExpresion() | Devuelve la expresión regular definida. |
| setFormatear(boolean formatear) | Impide que mientras se esté escribiendo sobre el componente se inserten caracteres que no cumplan con la expresión regular. |
| setEnumExpresiones(EExpresiones enum) | Permite seleccionar una de las expresiones definidas por defecto. |
| getEnumExpresiones() | Devuelve el enumerador de la expresión definida. |

Tabla 2: Descripción de la clase JFormateador

| Nombre de la clase: EExpresiones | |
|--|--|
| Descripción: Clase Enumerador que contiene nombres de expresiones que vienen definidas por defecto. | |
| Método | Descripción |
| EExpresiones(int valor) | Constructor el cual recibe por parámetro el número del valor del enumerador. |
| getValor() | Devuelve el valor del enum. |

Tabla 3: Descripción de la clase EExpresiones

| Nombre de la clase: Validar | |
|---|---|
| Descripción: Esta clase es usada en los componentes de entrada de texto y la función que ella realiza es que no permite que se copie en el componente texto del portapapeles que no cumpla con las características que requiera cada componente. | |
| Método | Descripción |
| Validar() | Constructor. |
| Validar(String expresionRegular) | Constructor que recibe por parámetros la expresión regular con la cual tiene que cumplir el texto del portapapeles para que pueda ser insertado en el componente. |
| addComponenteTexto(JTextComponent componteDeTexto) | Recibe el componente de texto al cual se le va a analizar el texto del portapapeles. |

| | |
|---------------------------------------|--|
| validar(JTextComponent componte) | Este método recibe por parámetro un componente de texto al cual le agrega las funciones de validación. |
| setExpresionRegular(String expresion) | Indica cual es la expresión regular con la cual tiene que cumplir el texto del portapapeles para que pueda ser insertado en el componente. |
| getExpresionRegular() | Devuelve la expresión regular definida. |

Tabla 4: Descripción de la clase Validar

2.6.2 Diagrama de Componentes

“El Diagrama de Componentes se usa para modelar la estructura del software, incluyendo las dependencias entre los componentes de software, los componentes de código binario, y los componentes ejecutables. En el Diagrama de Componentes se modela componentes del sistema, a veces agrupados por paquetes, y las dependencias que existen entre componentes (y paquetes de componentes).” (Agüero, 2002)

A continuación se muestra el diagrama de componentes de la librería:

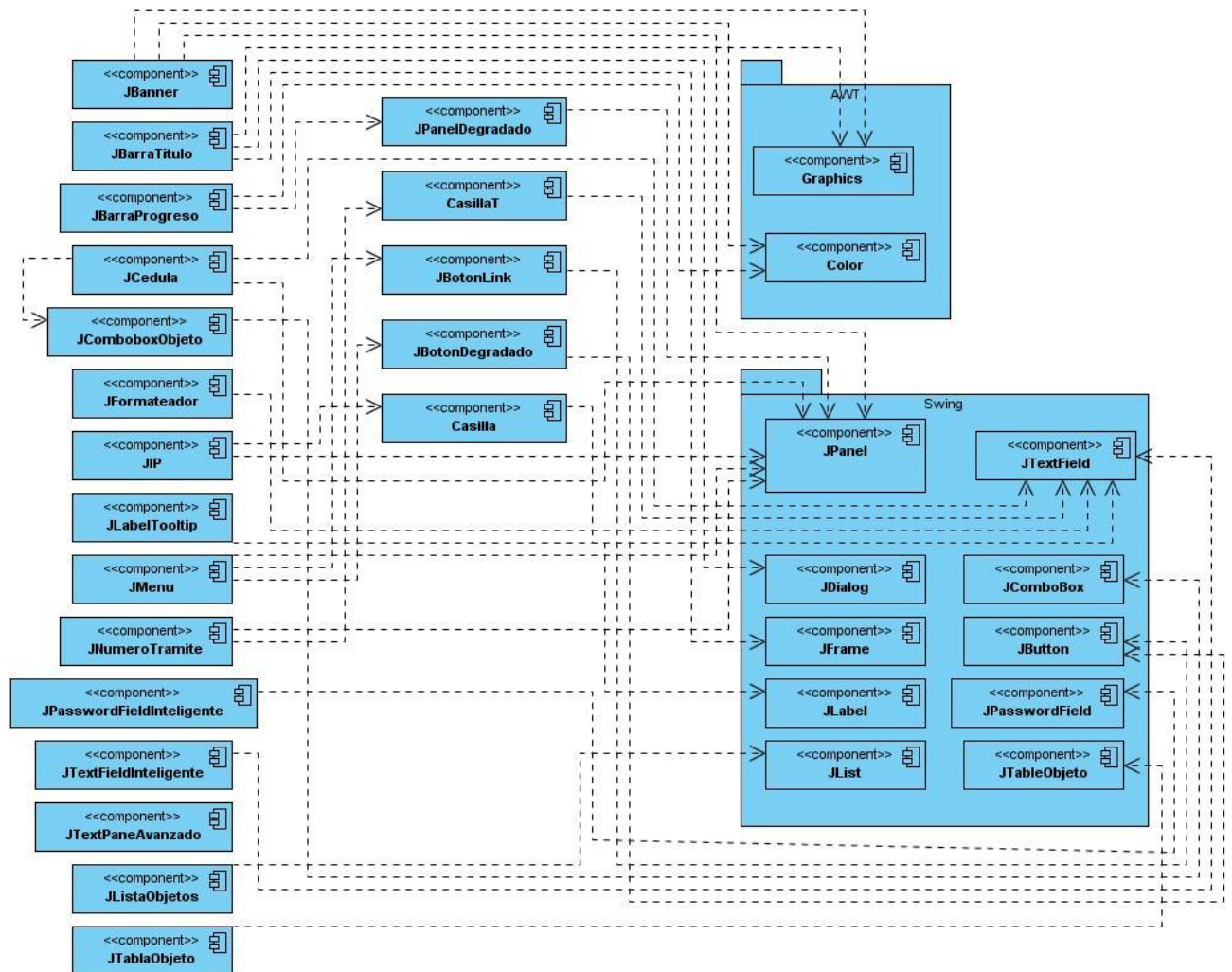


Fig.25 Diagrama de componentes de la librería.

2.7 Estándares de Codificación

Los estándares de codificación son un conjunto de reglas a seguir por los desarrolladores con el objetivo de establecer un orden y un formato común en el código fuente del software en desarrollo, a continuación se muestran los estándares aplicados durante la implementación de la librería de componentes.

2.7.1 Ficheros Fuente Java

Cada fichero fuente Java contiene una única clase o interfaz pública. Sólo algunas clases o interfaces privadas que estén asociadas a una clase pública, pueden ponerse en el mismo fichero que la clase pública. La clase o interfaz pública debe ser la primera clase o interfaz del fichero.

2.7.2 Declaraciones de Clases e Interfaces

A continuación se describe las partes de la declaración de una clase o interfaz, en el orden en que deberán aparecer:

- Sentencia *class* o *interface*.
- Comentario de implementación de la clase o interfaz si fuera necesario (*/* . . . */*): Este comentario debe contener cualquier información aplicable a toda la clase o interfaz que no era apropiada para estar en los comentarios de documentación de la clase o interfaz.
- Variables de clase (*static*): Primero las variables de clase *public*, después las *protected*, después las de nivel de paquete (sin modificador de acceso), y después las *private*.
- Variables de instancia: Primero las *public*, después las *protected*, después las de nivel de paquete (sin modificador de acceso), y después las *private*.
- Constructores.
- Métodos: Estos métodos se deben agrupar por funcionalidad más que por visibilidad o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos públicos de instancia. El objetivo es hacer el código más legible y comprensible.

2.7.3 Ruptura de Líneas

Cuando una línea de código sobrepasa los 80 caracteres se debe realizar un rompimiento de línea, este rompimiento deberá cumplir con los siguientes principios:

- Romper después de una coma.
- Romper antes de un operador.
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.

2.7.4 Comentarios

Los programas Java pueden tener dos tipos de comentarios: comentarios de implementación y comentarios de documentación.

- Los comentarios de documentación se limitan por `/**...*/`. Los comentarios de documentación son para describir la especificación del código, libre de una perspectiva de implementación, y para ser leídos por desarrolladores que pueden no tener el código fuente a mano.
- Los comentarios de implementación son delimitados por `/*...*/`, y `//`. Los comentarios de implementación son para comentar nuestro código o para comentarios acerca de una implementación particular.

2.7.5 Declaraciones

- Se recomienda una declaración por línea, ya que facilita los comentarios.
- Se deben inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir.
- Poner las declaraciones solo al principio de los bloques de código.

2.7.6 Declaraciones de Clases e Interfaces

Al codificar clases e interfaces, se siguen las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- La llave de cierre "}" empieza una nueva línea para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{".
- Los métodos se separan con una línea en blanco.

2.7.7 Sentencias

- Cada línea debe contener solo una sentencia.

2.7.8 Espacios en Blanco

Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre las secciones de un fichero fuente.
- Entre las definiciones de clases e interfaces.

Se debe usar siempre una línea en blanco en las siguientes circunstancias:

- Entre métodos.
- Entre las variables locales de un método y su primera sentencia.
- Antes de un comentario de bloque o de un comentario de una línea.
- Entre las distintas secciones lógicas de un método para facilitar la lectura.

Se deben usar espacios en blanco en las siguientes circunstancias:

- Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio.
- Debe aparecer un espacio en blanco después de cada coma en las listas de argumentos.
- Todos los operadores binarios excepto "." se deben separar de sus operandos con espacios en blanco. Los espacios en blanco no deben separar los operadores unarios, incremento "++" y decremento "--" de sus operandos.
- Las expresiones en una sentencia *for* se deben separar con espacios en blanco.
- Los casteos deben ir seguidos de un espacio en blanco.

2.7.9 Convenciones de Nombres

- Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas.
- Los nombres de las interfaces siguen la misma regla que las clases.
- Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que los forman en mayúscula.
- Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula.
- Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión "_".

(Hommel, 1999)

2.8 Conclusiones

En este capítulo se realizó una descripción de la arquitectura de la librería de componentes y se expusieron los principales objetivos por los cuales se decidió utilizar esta arquitectura. Se describieron además cada uno de los componentes de la librería, resaltando cuales son las funcionalidades de cada uno de ellos y su objetivo principal. Por último se expuso el diseño de la librería donde se describen cada

una de las clases que la conforman y además se muestran los diagramas de cada una de ellas así como los estándares de codificación utilizados a la hora de desarrollar la librería. Este capítulo sirve de ayuda a los desarrolladores que deseen utilizar esta librería a comprender sus funcionalidades así como su estructura.

3 CAPÍTULO 3: VALIDACIÓN DE LA PROPUESTA

3.1 Introducción

En el presente capítulo se abordarán las pruebas realizadas a cada uno de los componentes de la librería. Estas pruebas son realizadas con el objetivo de comprobar que todos los componentes cumplan con los requisitos funcionales y no funcionales propuestos. Las pruebas unitarias serán realizadas con el framework de prueba JUnit y con este se generará un reporte con los resultados de estas pruebas y serán analizados en este capítulo.

3.2 Pruebas Unitarias

Las pruebas unitarias son una actividad fundamental en XP, deben ser automatizadas y elaboradas por los desarrolladores antes y durante la implementación de un componente. Luego de terminada la implementación del mismo las pruebas unitarias deben ser todas correctas, asegurando así el buen funcionamiento del componente, es necesario que todas las pruebas sean correctas para poder integrarlo al sistema existente. (Malfará, y otros, 2006)

Para la realización de estas pruebas unitarias se utilizó el framework de prueba JUnit el cual viene integrado con el NetBeans. JUnit permite generar reportes en formato HTML que dan una vista de las pruebas realizadas al código. Estos reportes muestran información acerca de todos los métodos probados por el framework, de cada una de las clases de la librería.

El total de pruebas a la que fue sometida la librería es de 231, todas estas fueron ejecutadas en un tiempo de 7.401 milisegundos donde todas ellas devolvieron resultados exitosos. El resultado general se puede observar en la siguiente imagen:

Summary

| Tests | Failures | Errors | Success rate | Time |
|---------------------|----------|--------|--------------|-------|
| 231 | 0 | 0 | 100.00% | 7.401 |

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

| Name | Tests | Errors | Failures | Time(s) | Time Stamp | Host |
|---|-------|--------|----------|---------|---------------------|------------|
| cu.siqesap.cmp.ibanner | 12 | 0 | 0 | 0.452 | 2012-05-10T20:26:03 | fulgore-PC |
| cu.siqesap.cmp.ibarradetitulo | 15 | 0 | 0 | 0.466 | 2012-05-10T20:26:05 | fulgore-PC |
| cu.siqesap.cmp.ibarraprogreso | 14 | 0 | 0 | 0.470 | 2012-05-10T20:26:06 | fulgore-PC |
| cu.siqesap.cmp.icedula | 11 | 0 | 0 | 0.591 | 2012-05-10T20:26:07 | fulgore-PC |
| cu.siqesap.cmp.icomboboxobjeto | 14 | 0 | 0 | 0.481 | 2012-05-10T20:26:08 | fulgore-PC |
| cu.siqesap.cmp.ifromateador | 8 | 0 | 0 | 0.261 | 2012-05-10T20:26:10 | fulgore-PC |
| cu.siqesap.cmp.iip | 12 | 0 | 0 | 0.290 | 2012-05-10T20:26:11 | fulgore-PC |
| cu.siqesap.cmp.ilabeltooltip | 5 | 0 | 0 | 0.232 | 2012-05-10T20:26:12 | fulgore-PC |
| cu.siqesap.cmp.ilistaobjeto | 11 | 0 | 0 | 0.357 | 2012-05-10T20:26:13 | fulgore-PC |
| cu.siqesap.cmp.imenu | 44 | 0 | 0 | 1.565 | 2012-05-10T20:26:14 | fulgore-PC |
| cu.siqesap.cmp.inumerodetramite | 13 | 0 | 0 | 0.315 | 2012-05-10T20:26:20 | fulgore-PC |
| cu.siqesap.cmp.ipaneldegradado | 15 | 0 | 0 | 0.268 | 2012-05-10T20:26:21 | fulgore-PC |
| cu.siqesap.cmp.ipasswordinteligente | 6 | 0 | 0 | 0.255 | 2012-05-10T20:26:22 | fulgore-PC |
| cu.siqesap.cmp.itablaobjeto | 26 | 0 | 0 | 0.661 | 2012-05-10T20:26:23 | fulgore-PC |
| cu.siqesap.cmp.itexfieldinteligente | 6 | 0 | 0 | 0.155 | 2012-05-10T20:26:24 | fulgore-PC |
| cu.siqesap.cmp.itexpaneabansado | 19 | 0 | 0 | 0.582 | 2012-05-10T20:26:25 | fulgore-PC |

Fig.26 Resultado General

Los resultados de las pruebas unitarias realizadas a cada uno de los componentes de la librería se encuentran en los anexos del presente trabajo. A continuación se muestra un ejemplo las pruebas realizadas a uno de los componentes de la librería:

3.2.1 Prueba Unitaria JBanner

El componente JBanner fue comprobado en 4 iteraciones donde en cada una de ellas se le realizó al código del componente un total de 12 pruebas. En la primera iteración se encontraron 3 errores, en la segunda iteración se encontraron 2 errores, en la tercera iteración se encontró un error, ya en la cuarta iteración todos los errores encontrados anteriormente habían sido corregidos. Según se fueron realizando las iteraciones con las pruebas correspondientes se fueron corrigiendo los errores encontrados en cada una de estas. Como resultado se tiene que en la iteración final todas las pruebas fueron realizadas con éxito, en un tiempo de 0.452 milisegundos. Estos resultados se pueden apreciar en la siguiente tabla:

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

| Name | Tests | Errors | Failures | Time(s) | Time Stamp | Host |
|-----------------------------|-------|--------|----------|---------|---------------------|------------|
| JBannerTest | 12 | 0 | 0 | 0.452 | 2012-05-10T20:26:03 | fulgore-PC |

Tests

| Name | Status | Type | Time(s) |
|------------------------------|---------|------|---------|
| testGetColorDegradado1 | Success | | 0.094 |
| testSetColorDegradado1 | Success | | 0.003 |
| testGetColorDegradado2 | Success | | 0.001 |
| testSetColorDegradado2 | Success | | 0.071 |
| testGetColorFondo | Success | | 0.002 |
| testSetColorFondo | Success | | 0.002 |
| testGetImageIzquierda | Success | | 0.002 |
| testSetImageIzquierda | Success | | 0.001 |
| testGetImageDerecha | Success | | 0.011 |
| testSetImageDerecha | Success | | 0.001 |
| testGetjPanelDegradadoRadial | Success | | 0.001 |
| testSetjPanelDegradadoRadial | Success | | 0.080 |

Fig.27 Resultado de las pruebas unitarias de JBanner

3.3 Resumen de las Pruebas Unitarias

La siguiente tabla muestra la cantidad de errores que fueron encontrados en cada una de las iteraciones de prueba de cada componente de la librería y muestra también la cantidad de pruebas realizadas a cada componente por iteración:

| Componente | Iteración | Cantidad de Pruebas | Errores |
|------------|-----------|---------------------|---------|
| JBanner | 1 | 12 | 3 |
| | 2 | 12 | 2 |
| | 3 | 12 | 1 |
| | 4 | 12 | 0 |

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

| | | | |
|-----------------|---|----|---|
| JBarraTitulo | 1 | 12 | 2 |
| | 2 | 12 | 0 |
| JBarraProgreso | 1 | 14 | 3 |
| | 2 | 14 | 2 |
| | 3 | 14 | 1 |
| | 4 | 14 | 0 |
| JCedula | 1 | 11 | 2 |
| | 2 | 11 | 0 |
| JComboBoxObjeto | 1 | 14 | 5 |
| | 2 | 14 | 2 |
| | 3 | 14 | 1 |
| | 4 | 14 | 0 |
| JFormateador | 1 | 8 | 3 |
| | 2 | 8 | 0 |
| JIP | 1 | 12 | 3 |
| | 2 | 12 | 1 |
| | 3 | 12 | 0 |

Desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework para el desarrollo de la capa de presentación en aplicaciones de software de escritorio.

| | | | |
|-----------------------|---|----|---|
| JLabelTooltip | 1 | 5 | 3 |
| | 2 | 5 | 0 |
| JBotonLink | 1 | 1 | 1 |
| | 2 | 1 | 1 |
| | 3 | 1 | 0 |
| JPanelGrupo | 1 | 15 | 4 |
| | 2 | 15 | 3 |
| | 3 | 15 | 1 |
| | 4 | 15 | 0 |
| JBotonDegradado | 1 | 18 | 7 |
| | 2 | 18 | 5 |
| | 3 | 18 | 2 |
| | 4 | 18 | 0 |
| JPanelContenedor | 1 | 6 | 2 |
| | 2 | 6 | 0 |
| JScrollPaneContenedor | 1 | 4 | 0 |
| JNumeroTramite | 1 | 13 | 5 |

| | | | |
|---------------------------|---|----|----|
| | 2 | 13 | 3 |
| | 3 | 13 | 0 |
| JPanelDegradado | 1 | 15 | 1 |
| | 2 | 15 | 0 |
| JTextFieldInteligente | 1 | 6 | 0 |
| JPasswordFieldInteligente | 1 | 6 | 2 |
| | 2 | 6 | 0 |
| JTextPaneAvanzado | 1 | 19 | 7 |
| | 2 | 19 | 5 |
| | 3 | 19 | 4 |
| | 4 | 19 | 2 |
| | 5 | 19 | 0 |
| JTablaObjeto | 1 | 26 | 10 |
| | 2 | 26 | 8 |
| | 3 | 26 | 6 |
| | 4 | 26 | 5 |
| | 5 | 26 | 3 |

| | | | |
|--------------|---|----|---|
| | 6 | 26 | 1 |
| | 7 | 26 | 0 |
| JListaObjeto | 1 | 11 | 4 |
| | 2 | 11 | 2 |
| | 3 | 11 | 0 |

Tabla 5: Resumen de las pruebas unitarias

A continuación se muestra gráficamente un resumen promediado de los resultados de las pruebas unitarias realizadas a cada uno de los componentes de la librería. En la primera iteración el 67% de las pruebas realizadas eran insatisfactorias, en la segunda iteración el 34% las pruebas realizadas eran insatisfactorias y en la tercera iteración el 16% de las pruebas realizadas también eran insatisfactorias, ya en la cuarta iteración todas las pruebas fueron realizadas satisfactoriamente.

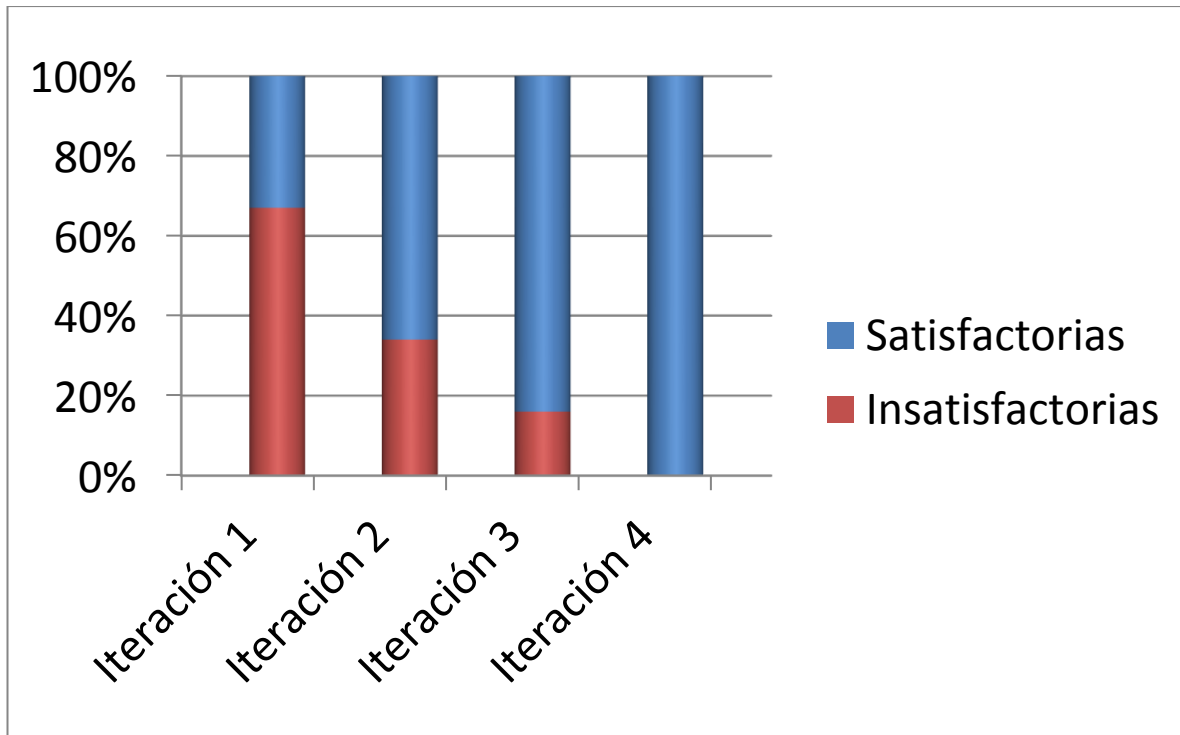


Fig.28 Resumen gráfico de las pruebas unitarias

3.4 Pruebas de Aceptación

Las pruebas de aceptación son pruebas definidas por el cliente y tienen como objetivo asegurar que las funcionalidades del sistema cumplan con lo que se espera de ellas. En efecto, las pruebas de aceptación corresponden a una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención. Las pruebas de aceptación tienen una importancia crítica para el éxito de una iteración. Por lo tanto el probador debe tenerlas creadas lo antes posible a partir del comienzo de la iteración, y lograr que el cliente las apruebe para poder presentárselas cuanto antes al equipo de desarrollo. (Malfará, y otros, 2006)

A cada uno de los componentes de la librería se le realizaron pruebas de aceptación. Estas pruebas son reflejadas mediante tablas que validan la realización de las mismas. Estas tablas están conformadas por seis parámetros que dan información acerca de la prueba realizada. Estos parámetros son:

- **Código:** Muestra un identificador para cada prueba realizada (normalmente se pone el nombre del componente seguido de las letras PA: pruebas de aceptación).
- **Nombre:** Indica el nombre de la prueba.
- **Descripción:** Se describe cual es la funcionalidad que se va a medir del componente al que se le esté realizando la prueba.
- **Condiciones de Ejecución:** Este parámetro indica cuales son las condiciones que se tienen que cumplir para que el componente realice correctamente la funcionalidad que se va a medir.
- **Resultados esperados:** Indica cual es el resultado que se obtendría de un correcto funcionamiento de la prueba.
- **Evaluación de la prueba:** Indica el estado de la prueba si es satisfactoria o insatisfactoria.

Las tablas de las pruebas de aceptación realizadas a cada uno de los componentes de la librería se encuentran en los anexos del presente trabajo. A continuación se muestra un ejemplo las pruebas realizadas a uno de los componentes de la librería:

3.4.1 Prueba de Aceptación JComboBoxObjeto

El componente JComboBoxObjeto fue comprobado en 3 iteraciones, en las dos primeras los resultados de las pruebas fueron insatisfactorios. Los errores presentes en este componente fueron corregidos y en la tercera iteración el resultado de la prueba fue satisfactorio. La siguiente tabla muestra la prueba de la tercera iteración:

| Caso de prueba de aceptación |
|--|
| Código: JComboBoxObjetoPA |
| Nombre: Mostrar listado de objetos |
| Descripción: Prueba la funcionalidad de mostrar un listado de objetos en el componente. |
| Condiciones de ejecución: <ul style="list-style-type: none"> • EL componente debe estar configurado previamente. • Se le debe insertar el listado de objetos al componente. |

| |
|---|
| Resultados esperados: El componente muestra en su lista desplegable las propiedades de los objetos que le fueron configuradas. |
| Evaluación de la prueba: Prueba satisfactoria. |

Tabla 6: Caso de prueba de aceptación de JComboBoxObjeto

3.5 Resumen de las Pruebas de Aceptación

La siguiente tabla muestra las evaluaciones de las pruebas que fueron encontradas en cada una de las iteraciones de prueba de cada componente de la librería:

| Componente | Nombre del Caso de Prueba | Iteración | Evaluación de la Prueba |
|-----------------|---------------------------|-----------|-------------------------|
| JBanner | Redimensionar ventana. | 1 | Prueba insatisfactoria |
| | | 2 | Prueba insatisfactoria |
| | | 3 | Prueba satisfactoria |
| JBarraTitulo | Mostrar título. | 1 | Prueba satisfactoria |
| JBarraProgreso | Mostrar progreso. | 1 | Prueba insatisfactoria |
| | | 2 | Prueba satisfactoria |
| JCedula | Insertar cédula. | 1 | Prueba satisfactoria |
| JComboBoxObjeto | Mostar listado de objetos | 1 | Prueba insatisfactoria |
| | | 2 | Prueba insatisfactoria |
| | | 3 | Prueba satisfactoria |
| JFormateador | Validar con expresiones | 1 | Prueba insatisfactoria |

| | | | |
|---------------------------|-----------------------------|---|------------------------|
| | regulares. | 2 | Prueba insatisfactoria |
| | | 3 | Prueba insatisfactoria |
| | | 4 | Prueba satisfactoria |
| JIP | Insertar dirección IP | 1 | Prueba satisfactoria |
| JLabelToolTip | Mostrar texto de ayuda. | 1 | Prueba satisfactoria |
| JMenu | Mostrar menú vertical. | 1 | Prueba insatisfactoria |
| | | 2 | Prueba insatisfactoria |
| | | 3 | Prueba satisfactoria |
| JNumeroTramite | Insertar número de trámite. | 1 | Prueba satisfactoria |
| JPanelDegradado | Mostrar degradado. | 1 | Prueba insatisfactoria |
| | | 2 | Prueba satisfactoria |
| JTextFieldInteligente | Insertar información. | 1 | Prueba satisfactoria |
| JPasswordFieldInteligente | Insertar contraseña. | 1 | Prueba satisfactoria |
| JTextPaneAvanzado | Mostrar texto con tokens. | 1 | Prueba insatisfactoria |
| | | 2 | Prueba satisfactoria |
| JTablaObjeto | Mostrar tabla de objetos | 1 | Prueba insatisfactoria |
| | | 2 | Prueba insatisfactoria |

| | | | |
|--------------|----------------------------|---|------------------------|
| | | 3 | Prueba satisfactoria |
| JListaObjeto | Mostrar listado de objetos | 1 | Prueba insatisfactoria |
| | | 2 | Prueba satisfactoria |

Tabla 7: Resumen de las pruebas de aceptación

A continuación se muestra gráficamente un resumen promediado de los resultados de las pruebas de aceptación realizadas a cada uno de los componentes de la librería. En la primera iteración el 9% de las pruebas realizadas eran insatisfactorias, en la segunda iteración el 5% las pruebas realizadas eran insatisfactorias y en la tercera iteración todas las pruebas fueron realizadas satisfactoriamente.

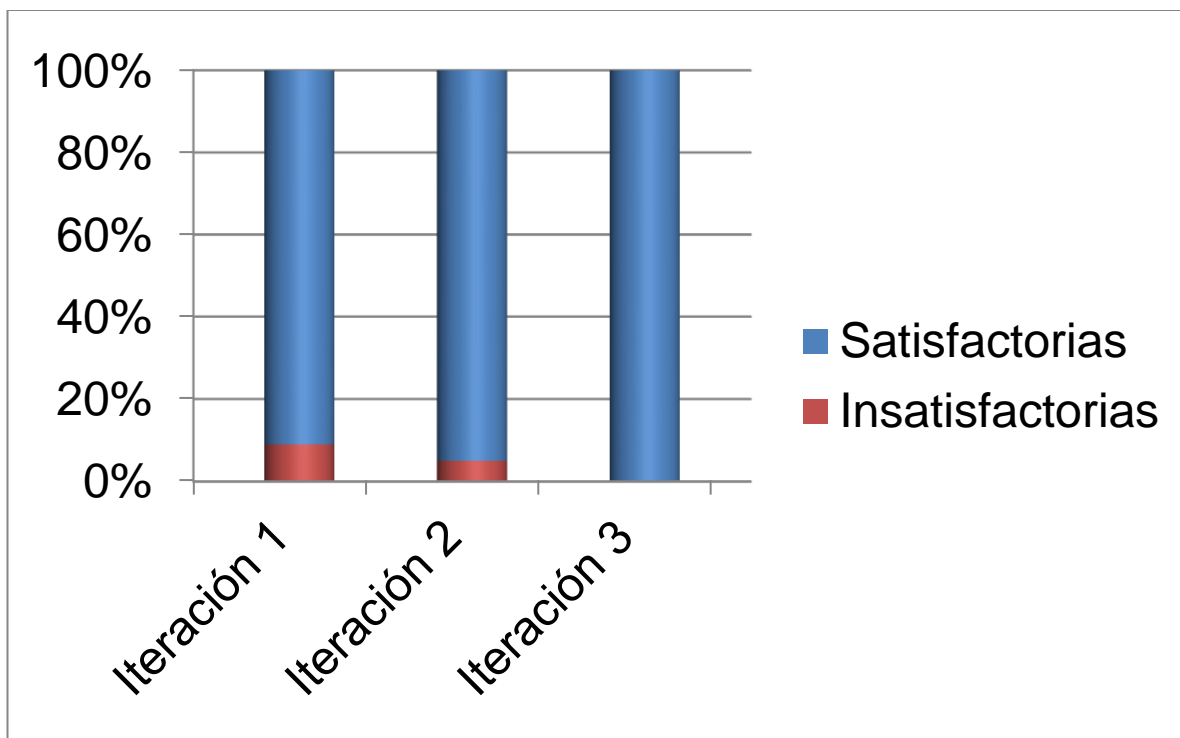


Fig.29 Resumen gráfico de las pruebas de aceptación

Independientemente de las pruebas que se le realizó a cada componente de la librería en el presente trabajo, estos fueron utilizados y puestos a prueba para el desarrollo de los sistemas SIGESAP y DigiDAP, que fueron aceptados y liberados por CaliSoft y actualmente se encuentran en uso en la División de Antecedentes Penales de Venezuela.

3.6 Utilización de los Componentes

A continuación se muestra una tabla que contiene la cantidad de veces que se utilizan cada uno de los componentes en el sistema SIGESAP:

| Componente | Cantidad de veces que se utilizó |
|-----------------------|----------------------------------|
| JBanner | 1 |
| JBarraTitulo | 5 |
| JBarraProgreso | 1 |
| JCedula | 10 |
| JComboBoxObjeto | 121 |
| JFormateador | 33 |
| JIP | 10 |
| JLabelToolTip | 110 |
| JMenu | 1 |
| JNumeroTramite | 15 |
| JPanelDegradado | 5 |
| JTextFieldInteligente | 1 |

| | |
|---------------------------|----|
| JPasswordFieldInteligente | 1 |
| JTextPaneAvanzado | 2 |
| JTablaObjeto | 93 |
| JListaObjeto | 5 |

Tabla 8: Cantidad de componentes en el sistema SIGESAP

3.7 Conclusiones

En este capítulo se expusieron las pruebas realizadas a cada uno de los componentes de la librería. Estas pruebas realizadas permitieron detectar errores no visibles que estaban presentes en la librería, y que posteriormente fueron corregidos. Los resultados de estas pruebas, tanto las unitarias como las de aceptación dan una constancia de la calidad del producto desarrollado que en este caso es la librería de componentes de interfaz de usuario.

CONCLUSIONES

A partir de los objetivos trazados referentes al desarrollo de una librería de componentes de interfaz gráfica de usuario basada en Swing Application Framework y AWT Framework y teniendo en cuenta los resultados esperados, se puede concluir que se cumplió con dichos objetivos de forma satisfactoria, evidenciándose los aspectos siguientes:

- Los marcos de trabajo Swing Application Framework y AWT Framework cuentan con una gran cantidad de componentes, pero sin embargo la mayoría de estos no contienen las características necesarias, por lo que no se ajustan a las necesidades de los desarrolladores y se hace engorroso el diseño de interfaces gráficas de usuarios para aplicaciones de escritorio basadas en Java.
- Se conformó una librería de componentes configurables y reutilizables, lo cual contribuirá a satisfacer las necesidades y expectativas de los desarrolladores, facilitando en gran medida el desarrollo de las aplicaciones que la utilicen.
- Por último, se validó la propuesta sometiendo cada uno de los componentes que conforman la librería a pruebas de aceptación y pruebas unitarias, las cuales fueron vencidas exitosamente, demostrando así, su buena calidad y correcto funcionamiento.

RECOMENDACIONES

Independientemente de que los objetivos trazados en el presente trabajo fueron cumplidos satisfactoriamente, surgieron nuevas ideas a lo largo del desarrollo de la librería que se proponen a continuación:

- Dotar la librería de nuevos componentes que faciliten el desarrollo de las interfaces de usuario en los centros de desarrollo de la UCI.
- Documentar la librería propuesta para facilitar su uso en los centros de desarrollo.
- Utilizar la librería propuesta en otros centros de desarrollo de la UCI.

BIBLIOGRAFÍA

De Nobrega, Maria. 2005. Blogia. [En línea] 4 de Junio de 2005. [Citado el: 5 de Enero de 2012.] http://curso_sin2.blogia.com/2005/060401-herramientas-case-rational-rose.-por-maria-de-nobrega.php.

López González, Isis Imelda y Rivera Carrera, Amalia Beatriz . 2009. *Sistema de control presupuestal.* 2009.

Agüero, Jorge. 2002. *Introducción a UML.* 2002.

Angelfire. 2011. Geet the planet. [En línea] 11 de Mayo de 2011. [Citado el: 5 de Junio de 2012.] <http://geektheplanet.net/5462/patrones-gof.xhtml#>.

Campo, Gustavo Damián . 2009. *Patrones de Diseño, Refactorización y Antipatrones.* 2009.

Carrasco Moñino, Manuel. 2010. Slideshare. [En línea] 07 de Mayo de 2010. [Citado el: 27 de 12 de 2011.] <http://www.slideshare.net/dodotis/gwt-iv-mvp>.

Carrillo Pérez, Isaías , Pérez González, Rodrigo y Rodríguez Martín, Aureliano David . 2008. *Metodología de desarrollo del software.* 2008.

Casal Terreros, Julio . 2005. Msdn. [En línea] 12 de Octubre de 2005. [Citado el: 10 de Enero de 2012.] <http://msdn.microsoft.com/es-es/library/bb972268.aspx#mainSection>.

de la Torre LLorente, César y Calvarro Nelson, Javier. 2010. *Guía de Arquitectura N-Capas orientada al Dominio con .Net 4.0.* 2010.

Doar, M.B. 2005. *Practical Development Environments.* 2005.

Econ. Félix Murillo Alfaro. 1999. *Herramientas Case.* 1999.

García de Jalón, Javier y Rodríguez, José Ignacio. 2000. *Aprenda Java como si estuviera en primero.* 2000.

Gastón. 2008. clubdesarrolladores. [En línea] 2 de Septiembre de 2008. [Citado el: 15 de Febrero de 2012.] <http://www.clubdesarrolladores.com/articulos/mostrar/63-metodologia-scrum/2>.

Gómes Gallego, Juan Pablo y Galves, Jorge. 2007. *Fundamentos de la metodología RUP.* 2007.

Green, Dale . 2004. Universitat Paderborn. [En línea] 2004. [Citado el: 15 de Diciembre de 2011.] <http://groups.uni-paderborn.de/cc/tools/tutorial/reflect/index.html>.

Hernández, Pedro. 2009. *JUnit*. 2009.

Hommel, Scott. 1999. *Convenciones de Código para el lenguaje de programación JAVA*. 1999.

Itera. 2012. Itera. [En línea] 2012. [Citado el: 10 de Enero de 2012.] http://www.iteraprocess.com/index.php?option=com_content&task=view&id=18&Itemid=42&limit=1&limitstart=3.

Jiménez Rodá, Javier. 2008. Patterns and Practices. [En línea] 16 de Diciembre de 2008. [Citado el: 17 de Junio de 2012.] <http://jimenezroda.wordpress.com/2008/12/16/low-coupling-pattern-patron-bajo-acoplamiento/>.

Malfará, Dayvis , y otros. 2006. *Testing en eXtreme Programming*. 2006.

Manzanedo del Campo, Miguel Ángel . 1999. Guía de Iniciación al Lenguaje JAVA. [En línea] Octubre de 1999. [Citado el: 15 de Diciembre de 2011.] http://zarza.usal.es/~fgarcia/doc/tuto2/IV_3.htm.

Martínez, David . 2002. Hackerdude. [En línea] 18 de Octubre de 2002. [Citado el: 10 de Enero de 2012.] <http://www.hackerdude.com/2002/10/18/programacion-extrema-mini-introduccion/>.

Mastermagazine. 2004. Mastermagazine. *Mmastermagazine*. [En línea] 2004. <http://www.mastermagazine.info/termino/3868.php>.

Miss, Alexander . Scribd. [En línea] [Citado el: 12 de Diciembre de 2011.] <http://es.scribd.com/doc/78605203/Visual-Paradigm>.

Navarrete, Rosa. 2006. *Escuela Politécnica Nacional*. 2006.

NetBeans. 2010. NetBeans. [En línea] 2010. [Citado el: 15 de Febrero de 2012.] http://netbeans.org/index_es.html.

Oracle. 2010. Trail: The Reflection API. [En línea] 2010. [Citado el: 15 de Febrero de 2012.] <http://docs.oracle.com/javase/tutorial/reflect/>.

Pressman, Roger S. 2007. *Ingeniería del Software*. 2007.

Ramos Fernández, Pablo . 2006. *El Patrón Fachada*. 2006.

Retamar, Angel . 2009. *Mi primera hora con Eclipse.* 2009.

Reynoso , Carlos y Kiccillof, Nicolás . 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

Saavedra Gutierrez , Jorge A. 2007. El Mundo Informático. [En línea] 8 de Mayo de 2007. [Citado el: 18 de Junio de 2012.]

Sánchez Asenjó, Jorge. 2004. *Java2.* 2004.

Scribd. 2011. Scribd. [En línea] 19 de Junio de 2011. [Citado el: 5 de Enero de 2012.] <http://es.scribd.com/doc/59698939/Programacion-Extrema>.

Sgoliver. 2009. Sgoliver. [En línea] 12 de Febrero de 2009. [Citado el: 2 de Febrero de 2012.] <http://www.sgoliver.net/blog/?p=451>.

Sierra, Daniel . 2007. Slideshare. [En línea] 15 de Noviembre de 2007. [Citado el: 15 de Enero de 2012.] <http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml>.

Torazana, Ivon y Gómez, Oriana. 2005. [En línea] 2005. [Citado el: 15 de Diciembre de 2011.] http://es.scribd.com/cesarmarcano_ve/d/8962964-uml.

Zukowski, John. 2003. *Programación Java2 J2SE1.4.* 2003.