

# *Universidad de las Ciencias Informáticas*



## *Facultad 3*

*Título: Biblioteca de componentes de interfaz de usuario para aplicaciones de escritorio desarrolladas en Java.*

*Trabajo de diploma para optar por el título de  
Ingeniero en Ciencias Informáticas*

**Autores:** Greter Gómez Martínez  
Anchel Durán Bolaño

**Tutores:** Ing. Reinier Silverio Figueroa  
Msc. Danaysa Macías Hernández

**Co-tutora:** Ing. Nailin Pérez Martínez

**La Habana, Cuba  
2015**

## DECLARACIÓN JURADA DE AUTORÍA

Declaramos por este medio que nosotros, Greter Gómez Martínez y Anchel Durán Bolaño somos los autores del trabajo *Biblioteca de componentes de interfaz de usuario para aplicaciones de escritorio desarrolladas en Java* y autorizamos a la Universidad de las Ciencias Informáticas hacer uso de la investigación en su beneficio, así como los derechos patrimoniales, con carácter exclusivo.

Para que así conste se firma la presente en La Habana a los \_\_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

---

Greter Gómez Martínez

(Autor)

---

Anchel Durán Bolaño

(Autor)

---

Ing. Reinier Silverio Figueroa

(Tutor)

---

Msc. Danaysa Macías Hernández

(Tutora)

---

Ing. Nailin Pérez Martínez

(Co-tutora)

## **RESUMEN**

La Universidad de las Ciencias Informáticas tiene como propósito la creación de aplicaciones y servicios. Para ello se han creado centros de desarrollo como CEGEL, perteneciente a la Facultad 3. CEGEL lo integran diferentes proyectos productivos dentro de los que se encuentran SIGFE, SIRECC Y SIGESAP, los cuales para el desarrollo de sus aplicaciones utilizan el marco de trabajo XEGFORT. Este marco lleva incorporado una biblioteca de componentes que presenta deficiencias demorando el tiempo de desarrollo de las interfaces visuales, provocando además, retraso en la entrega del producto final. Por lo que se hace necesario disminuir el tiempo de desarrollo de las interfaces visuales de las aplicaciones que utilizan el marco de trabajo XEGFORT. Para ello se propone desarrollar una biblioteca de componentes gráficas para aplicaciones de escritorio desarrolladas en Java, mediante las herramientas de JavaFX. Esta tecnología es una plataforma de software que permite crear y ejecutar aplicaciones que pueden aplicarse en variedades de dispositivos. Dado que la plataforma JavaFX está implementada en Java, los desarrolladores de los centros que utilizan XEGFORT pueden aprovechar sus habilidades para desarrollar las interfaces visuales, utilizando la biblioteca de componentes. El presente trabajo expone un estudio de varias bibliotecas de interfaz gráfica, se muestra la selección de las herramientas, tecnologías y se brinda una descripción de la solución de software. Se exponen además, las disciplinas de la metodología de desarrollo y se realizó la validación de la solución mediante técnicas y métricas.

**Palabras claves:** aplicaciones, biblioteca, componentes, dispositivos, plataforma, XEGFORT.

# ÍNDICE DE CONTENIDO

INTRODUCCIÓN .....	1
FUNDAMENTACIÓN TEÓRICA.....	7
1.1 Bibliotecas de componentes de interfaz de usuario en Java .....	7
1.2 Metodología para el desarrollo del software.....	11
1.3 Arquitecturas de software .....	12
1.4 Patrones de Diseño .....	14
1.5 Especificación de Java Beans.....	15
1.6 Especificación de JavaFX Beans .....	16
1.7 Tecnologías .....	17
1.8 Herramientas .....	20
1.9 Técnicas y Métricas .....	23
1.10 Pruebas .....	25
Conclusiones Parciales.....	27
DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN.....	28
2.1 Modelado de negocio.....	28
2.2 Requisitos.....	31
2.3 Diseño .....	36
2.4 Implementación .....	41
Conclusiones Parciales.....	50
VALIDACIÓN DE LA INVESTIGACIÓN.....	51
3.1 Técnicas y métricas para validar los requisitos y el diseño .....	51
3.2 Verificación de la implementación.....	54
3.3 Validación mediante caso de estudio .....	57
3.4 Validación de las variables de investigación .....	66
Conclusiones Parciales.....	73
CONCLUSIONES GENERALES .....	74
RECOMENDACIONES .....	74
BIBLIOGRAFÍA .....	75

## INTRODUCCIÓN

Las tecnologías de información desempeñan un papel importante en el desarrollo de sistemas informáticos y ligados a ellos la evolución de nuevas tecnologías. La Universidad de las Ciencias Informáticas (UCI), se caracteriza por producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación (UCI, 2012). Debido a ello se han creado centros de desarrollo, con el objetivo de obtener soluciones informáticas para las cuales en su mayoría es muy común la utilización de marcos de trabajo.

El Centro de Gobierno Electrónico (CEGEL), es uno de los centros de desarrollo creados en la UCI, tiene como objetivo satisfacer necesidades de entidades y organizaciones mediante el desarrollo de productos, servicios y soluciones informáticas para la gestión de las áreas de gobierno (CEGEL, 2006). El centro CEGEL para la elaboración de sus productos desarrolló el marco de trabajo XEGFORT basándose en la tecnología Java *Enterprise Edition* (J2EE<sup>1</sup> por sus siglas en inglés), como plataforma para el desarrollo de aplicaciones empresariales. XEGFORT se creó con el objetivo de implementar los sistemas registrales que se desarrollan en este centro. Para el trabajo en las interfaces gráficas, XEGFORT incorpora una biblioteca de componentes<sup>2</sup> basada en Swing como una Interfaz de Programación de Aplicaciones (API) para el desarrollo de componentes visuales.

La biblioteca de componentes de interfaz de usuario que presenta XEGFORT se creó sobre la plataforma Java *Standard Edition* (JavaSE<sup>3</sup> por sus siglas en inglés). Esta biblioteca contiene código y datos que proporcionan servicios a programas independientes que permiten gestionar la información, proporcionando herramientas así como programas y librerías para facilitar el desarrollo de las interfaces visuales. Los componentes contenidos en esta biblioteca, permiten gestionar la información entre la aplicación informática y los usuarios, manteniendo el enfoque orientado a objetos hasta las interfaces de usuarios (Frías, 2012).

En la entrevista (Ver Anexo 1) realizada a especialistas en el desarrollo de sistemas registrales, pertenecientes a los proyectos SIGFE<sup>4</sup>, SIRECC<sup>5</sup> y SIGESAP<sup>6</sup>, con experiencia en el uso de XEGFORT, se identificaron deficiencias en el funcionamiento del marco de trabajo. Estas se encuentran en el desarrollo de interfaces visuales utilizando los componentes que brinda la biblioteca gráfica de

---

<sup>1</sup> Esta categoría agrupa todas las tecnologías que proporciona Java para la implementación de aplicaciones empresariales.

<sup>2</sup> Una unidad de composición con interfaces especificadas de manera contractual y dependencias de contexto completamente explícitas. Un componente de software puede desplegarse independientemente y es sujeto de composición por parte de terceros.

<sup>3</sup> Dentro de esta categoría se aglomeran todas las tecnologías existentes en Java para sistemas comunes o estándares.

<sup>4</sup> Sistema de Gestión de Ferias y Eventos.

<sup>5</sup> Sistema de Informatización Registral de la Cámara de Comercio.

<sup>6</sup> Sistema de Gestión de Antecedentes Penales.

XEGFORT, empleando tiempo adicional para conectar las acciones de un flujo de trabajo con los componentes de la interfaz.

Entre las deficiencias que presenta la biblioteca se encuentra la falta de sincronización entre la información contenida en el modelo<sup>7</sup> y lo que muestra el delegado<sup>8</sup> o vista al usuario. Lo que consiste en la necesidad constante de mantener a través de la programación la vista actualizada con cada cambio que se realiza en la información. Los eventos del modelo presentan escasa implementación provocando que no todos los eventos que están sujetos a cambios como de selección o edición sean escuchados, trayendo consigo necesidad de implementaciones adicionales y código repetitivo. Existen funciones nativas de determinados componentes que no trabajan acorde a sus especificaciones, implicando la necesidad de re-implementarlas e implementar funcionalidades adicionales que logren la función.

Los elementos que conforman el flujo de pasos del menú no admiten agrupaciones por jerarquías que respondan a las necesidades de representarlos de acuerdo al negocio, trayendo como consecuencia un incorrecto orden en la secuencia del menú. Los botones utilizados para navegar a través de los diferentes pasos de un flujo de trabajo se configuran al inicio del flujo. Esto limita la posibilidad de cambiar parámetros como: nombre, visibilidad y disponibilidad, en tiempo de ejecución, lo que conlleva a la implementación de validaciones que hacen poco intuitivo el trabajo del usuario final en la aplicación. Además, el menú lateral no es desplegable, causando que el área de trabajo cuente con un espacio limitado para mostrar los componentes de interfaz gráfica.

Los resultados obtenidos a partir de la entrevista realizada, permiten deducir que el tiempo de desarrollo e implementación de las interfaces visuales utilizando la biblioteca de interfaz gráfica de XEGFORT, se afecta debido a las deficiencias de los componentes antes mencionados. Pues no facilitan una adecuada interacción entre las propiedades de sus funciones y las necesidades del usuario para gestionar la información que muestran o capturan los componentes. Influyendo en la desorientación del usuario, confusión y falta de iniciativa para interactuar con la aplicación.

Se estima que el desarrollo de las interfaces se encuentra en intervalos de tiempo de dos a ocho horas, resultado obtenido a partir de la entrevista realizada a los especialistas en el desarrollo de sistemas registrales del centro. Para este proceso se tienen en cuenta varios aspectos como son las interfaces visuales simples y las interfaces visuales complejas. Las interfaces complejas invierten dos o más horas de tiempo en su desarrollo que las interfaces simples. Estas exigen una mayor gestión de la

---

<sup>7</sup> El modelo es responsable de mantener la información sobre el estado del componente.

<sup>8</sup> El delegado de interfaz de usuario es responsable de mantener la información sobre cómo dibujar el componente en la pantalla.

información, debido a la cantidad de componentes que contiene la interfaz para la gestión de los datos. Lo cual provoca una duplicación del intervalo de tiempo que demora el desarrollo e implementación de las interfaces visuales.

También se tiene en cuenta el nivel de experiencia de los especialistas que interactúan en el desarrollo de las aplicaciones que utilizan XEGFORT. Mientras menor experiencia tengan en el lenguaje de programación mayor es el tiempo de demora para desarrollar la interfaz, provocando un incremento del tiempo de hasta tres horas por cada interfaz que se desarrolle.

Estos aspectos que intervienen negativamente en el diseño visual de interfaces gráficas, provocan a largo plazo un promedio de pérdida de tiempo en un 35% al 45% del tiempo destinado al producto. Puede haberse empleado mucho en el desarrollo de una aplicación, pero se estaría fallando si los usuarios finales no encuentran un entorno fácil, rápido e intuitivo con el cual operar para conseguir sus metas (SublimeSolutions, 2010).

Debido a las dificultades antes mencionadas se hizo necesario buscar una alternativa que solucione las deficiencias encontradas en la biblioteca de componentes que utiliza XEGFORT.

Por lo que el **problema a resolver** queda formulado de la siguiente manera: ¿Cómo disminuir el tiempo de desarrollo de las interfaces visuales de las aplicaciones que utilizan el marco de trabajo XEGFORT?

Se define como **objeto de estudio**: Bibliotecas de componentes gráficos para aplicaciones de escritorio desarrolladas en Java.

Para dar respuesta al problema planteado se trazó el siguiente **objetivo general**: Desarrollar una biblioteca de componentes gráficos para aplicaciones de escritorio en Java, que permita disminuir el tiempo de desarrollo de las interfaces visuales de las aplicaciones que hacen uso del marco de trabajo XEGFORT.

Inciendo en el **campo de acción**: Bibliotecas de componentes gráficos de XEGFORT.

A partir del cual se desglosan los siguientes **objetivos específicos**:

- ✓ Elaborar el marco teórico de la investigación para conocer las principales tendencias en el desarrollo de bibliotecas de componentes gráficos en aplicaciones Java.
- ✓ Implementar la biblioteca de componentes gráficos para el desarrollo de interfaces visuales en XEGFORT, que contribuya a disminuir las deficiencias existentes.

- ✓ Validar la solución desarrollada a partir de la ejecución de técnicas y métricas aplicables al desarrollo de la solución, que permitan evaluar los resultados.

Estableciendo como **idea a defender**: Con el desarrollo de la biblioteca de componentes gráficos para aplicaciones de escritorio desarrolladas en Java, se disminuirá el tiempo de desarrollo de las interfaces visuales de las aplicaciones que hacen uso del marco de trabajo XEGFORT.

Para cumplir con los objetivos específicos se planificaron las siguientes **tareas de la investigación**:

1. Estudio del estándar Especificación de Java *Beans*, para fundamentar los conocimientos del desarrollo de componentes de interfaz gráfica en Java.
2. Estudio de la arquitectura de la biblioteca gráfica Swing, para fundamentar los conocimientos sobre la implementación de los componentes de interfaz gráfica.
3. Estudio del estándar Especificación de JavaFX *Beans*, para fundamentar los conocimientos del desarrollo de componentes de interfaz gráfica utilizando la tecnología JavaFX.
4. Estudio de la arquitectura de JavaFX para comprender el funcionamiento de la tecnología.
5. Estudio de la herramienta de diseño JavaFX *Scene Builder 2.0*, para comprender el proceso de desarrollo de las interfaces gráficas en JavaFX.
6. Estudio de mecanismos de empaquetado de los componentes de interfaz gráfica, para fundamentar los conocimientos acerca del almacenamiento de los eventos, operaciones y funcionalidades.
7. Estudio del lenguaje de codificación FXML, para fundamentar los conocimientos acerca de la construcción de objetos gráficos en JavaFX.
8. Estudio de mecanismos de inclusión de componentes en la herramienta de diseño JavaFX *Scene Builder 2.0* para su utilización en el desarrollo de interfaces gráficas.
9. Identificación de los componentes a incluir en la biblioteca de interfaz gráfica de usuario.
10. Descripción de los componentes de interfaz gráfica identificados.
11. Diseño de los componentes de interfaz gráfica descritos.
12. Implementación de los diseños modelados.
13. Empaquetado de los componentes de interfaz gráfica implementados.



14. Inclusión de los componentes de interfaz gráfica empaquetados a la paleta de componentes de JavaFX *Scene Builder 2.0*.
15. Prueba de los componentes de interfaz gráfica incluidos en la paleta de componentes de JavaFX *Scene Builder 2.0*.

Para apoyar el desarrollo de la investigación se emplean los siguientes **métodos científicos**:

#### **Métodos teóricos:**

- ✓ **Análítico-Sintético:** se utilizó para realizar una investigación previa para arribar a conclusiones acerca del funcionamiento y conformación de las bibliotecas de componentes de interfaz gráfica. Permitiendo identificar y analizar documentos, bibliografías y conceptos necesarios para el desarrollo de la solución.
- ✓ **Histórico-Lógico:** se utilizó para realizar la investigación acerca de las principales bibliotecas de interfaz gráficas en Java desde su inicio como lenguaje de programación hasta la actualidad.
- ✓ **Modelación:** este método permitió la creación de diagramas para expresar información de manera organizada. En el proceso de diseño el modelado se convierte en una herramienta indispensable para el entendimiento de las diferentes etapas del desarrollo del software, permitiendo crear abstracciones del producto final.

#### **Métodos empíricos:**

- ✓ **Entrevista:** permitió obtener información referente a las experiencias con el uso del marco de trabajo XEGFORT, así como las principales deficiencias en el desarrollo de las interfaces gráficas y en la actual biblioteca de componentes que utiliza.
- ✓ **Medición:** permitió la aplicación de métricas y estándares para obtener información cuantitativa acerca de las propiedades o cualidades de los elementos de estudio.
- ✓ **Implementación:** permitió demostrar que la solución tiene ciertas propiedades o que bajo ciertas condiciones se comporta de una manera específica. Se empleó en la realización de la implementación de todo el diseño que se modeló anteriormente y así dejar plasmado las ventajas obtenidas.
- ✓ **Caso de Estudio:** se empleó con el objetivo de probar la validez de la solución mediante una interfaz visual que muestra vistas de cada uno de los componentes, permitiendo obtener resultados medibles. Permitted comprobar que se cumplieran las funcionalidades con las que deben cumplir los componentes de la biblioteca de interfaz gráfica.

Se define la siguiente **estructura del documento**, quedando conformado en tres capítulos:

### **Capítulo 1: Fundamentación teórica**

En este capítulo se aborda el estado del arte referente a la investigación y el marco de los conocimientos acerca de las herramientas a utilizar. Se realiza una síntesis en relación a las bibliotecas de componentes de interfaz de usuario en Java, así como la metodología a seguir y la arquitectura de software. Se analizan las investigaciones necesarias para mostrar la teoría relacionada con el lenguaje de programación JavaFX y la creación de los componentes utilizando esta tecnología. Se conceptualizan elementos importantes para la captura y validación de los requisitos así como los distintos patrones de diseño, las herramientas y tecnologías a utilizar para el diseño e implementación de la biblioteca de componentes.

### **Capítulo 2: Diseño e Implementación de la solución**

En este capítulo se describe la biblioteca de componentes como propuesta de solución a desarrollar y el proceso de informatización. Se aborda acerca de la descripción de la biblioteca de interfaz gráfica mediante el diagrama del modelo conceptual. Se hace referencia a las técnicas empleadas tanto en la captura como en la validación de los requisitos. Se presentan los requisitos funcionales y no funcionales con los que debe cumplir el software. Se muestra el diagrama de clases de uno de los componentes ejemplificando el modelo de clases con los que cumplen los componentes y el diagrama de componentes de la biblioteca de interfaz gráfica. Se explica la utilización de los patrones de diseño a utilizar durante la implementación. Además, se presentan los componentes de interfaz gráfica utilizando la tecnología JavaFX. Como resultados, se obtienen un conjunto de artefactos establecidos por la metodología de desarrollo propuesta en el Capítulo 1.

### **Capítulo 3: Validación de la Investigación**

En este capítulo se presenta la validación de los requisitos mediante técnicas y métricas. Se valida el diseño de la solución y la implementación, a través de métricas y métodos de prueba de caja blanca y caja negra específicamente haciendo uso de las técnicas de camino básico y partición equivalente respectivamente. Se hace uso del método de caso de estudio para verificar el funcionamiento de la biblioteca de componentes de interfaz gráfica. Además, se realiza la validación de las variables de investigación para validar la solución.

# 1

## FUNDAMENTACIÓN TEÓRICA

*"Creemos que el lenguaje de programación Java es un lenguaje maduro, listo para su uso generalizado.*

*No obstante, se espera alguna evolución de la lengua en los años por venir.*

*Tenemos la intención de gestionar esta evolución de una manera que es totalmente compatible con las aplicaciones existentes."*

*James Gosling, Bill Joy y Guy Steele*

### Introducción

El presente capítulo se enfoca en la realización de un análisis de las fuentes bibliográficas para obtener información actualizada acerca de la teoría necesaria para la investigación. Se describe el marco teórico de las principales bibliotecas de componentes existentes a nivel mundial y nacional. Se exponen los elementos relevantes de la metodología y la arquitectura por la que se guiará el desarrollo del trabajo. Se realiza un análisis de los patrones de diseño, tecnologías, lenguajes de programación y herramientas a utilizar. Se aborda sobre las técnicas y métricas para evaluar tanto requisitos como diseño, además de las pruebas para garantizar la efectividad del proceso de desarrollo y calidad del producto final.

### 1.1 Bibliotecas de componentes de interfaz de usuario en Java

En ciencias de la computación, una biblioteca de componentes es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas de componentes de interfaz de usuario contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos, permitiendo que el código y los datos se compartan y puedan modificarse de forma modular (Palma, 2011).

Estas presentan diferentes particularidades, por lo que se propone un estudio de sus principales características para obtener información acerca de sus funcionalidades. Entre las principales bibliotecas a ser analizadas se encuentran *Abstract Windows Toolkit* (AWT por sus siglas en inglés), Swing, SwingX y la biblioteca de componentes de interfaz gráfica de XEGFORT.

#### AWT

Es una biblioteca para el diseño de interfaces gráficas de usuario. Constituye un conjunto de clases que intentan ofrecer al programador funcionalidades para la creación de interfaces de usuario para cualquier aplicación en Java. El uso de AWT crea algunas limitaciones debido a que algunos de sus componentes no funcionan en todas las plataformas. Presenta uso excesivo de consumo de recursos del sistema operativo, además del tiempo en que demora desarrollar las interfaces visuales debido a

la falta de componentes avanzados (Sáez, 2010). Estos problemas dieron paso al surgimiento de Swing para la creación de interfaces de usuario en Java, la cual está construido sobre la tecnología AWT.

### **Swing**

Se creó para solucionar muchas de las limitaciones de AWT. Es flexible y presenta un potente juego de herramientas de Interfaz Gráfica de Usuario (GUI). Swing es una plataforma independiente, utiliza como diseño fundamental para cada componente el patrón arquitectónico Modelo-Vista-Controlador (MVC). Aporta más componentes que AWT, presenta una mejor apariencia a la solución de software y no depende de la plataforma donde se ejecuta para manejar sus funcionalidades. Es una herramienta potente, flexible, usa tablas y otros componentes complejos, y completamente adaptable a las necesidades de los clientes (Sánchez, 2004).

Esto no significa que AWT sea remplazada, simplemente se reutilizan sus funcionalidades en las nuevas capacidades de Swing. Sin embargo la interacción entre los componentes de esta biblioteca y el resto de las capas de una aplicación, se vuelve engorrosa debido a la complejidad de las interfaces que proporcionan. A pesar de ello el tiempo de ejecución de desarrollo de una aplicación es más ágil que en AWT (Sáez, 2010). Como una evolución de esta biblioteca se creó SwingX que está basada en una extensión del propio Swing.

### **SwingX**

Esta nueva biblioteca añade varias funciones no presentes en las anteriores bibliotecas, 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; 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 biblioteca SwingX es que presenta poca documentación, ocasionando que no muchos desarrolladores decidan utilizarla, debido a que es difícil de usar sus nuevas funcionalidades y componentes sin una documentación previa (Frías, 2012).

Además, de las bibliotecas antes descritas se hace necesario el estudio de la biblioteca de componentes de interfaz gráfica que utiliza el marco de trabajo XEGFORT. Para conocer sus principales características.

### **Biblioteca de componentes de interfaz gráfica de XEGFORT**

La biblioteca de componentes que presenta el marco de trabajo XEGFORT está basada en Swing y SwingX para el diseño y trabajo con las interfaces gráficas, para eliminar los problemas existentes en

el desarrollo de las interfaces de usuario para la plataforma JavaSE. Como diseño utilizó la arquitectura basada en componentes, lo que le permitió reutilizar componentes desarrollados anteriormente, sumándole un incremento considerable en la mantenibilidad<sup>9</sup> del código (Frías, 2012).

Esta biblioteca presenta deficiencias en sus principales componentes como la falta de sincronización entre el modelo y el delegado, los eventos del modelo presentan escasa implementación. Algunas funciones nativas de determinados componentes no trabajan acorde a sus especificaciones, los elementos del menú no admiten agrupaciones por jerarquía que respondan a las necesidades de representarlos de acuerdo al negocio. Además, se le pudieran incorporar algunos componentes para una mejor implementación entre las acciones de un flujo de trabajo y los componentes de la interfaz gráfica, permitiéndole reducir el tiempo de desarrollo de las aplicaciones.

### Valoración de las bibliotecas de interfaz gráfica estudiadas

A partir del estudio de las principales bibliotecas de componentes de interfaz gráfica se establece un análisis comparativo para obtener información de sus características. Para realizar este análisis se establece un grupo de indicadores a cumplir: multiplataforma, sincronización entre el modelo y el delegado, complejidad de las interfaces, optimización del código y el tiempo empleado en implementaciones adicionales. El análisis de las bibliografías de las bibliotecas de interfaz gráfica antes estudiadas, permitió definir los indicadores con el objetivo de erradicar los problemas encontrados para la solución. A continuación se muestra la tabla comparativa:

Bibliotecas	AWT	Swing	SwingX	BCG XEGFORT
Multiplataforma	No	Sí	Sí	Sí
Sincronización entre el modelo y el delegado	No	Sí	Sí	No
Complejidad de las interfaces	Sí	Sí	Sí	No
Optimización del código	No	Sí	Sí	Sí
Tiempo empleado en implementaciones adicionales	Sí	Sí	Sí	Sí

**Tabla 1. Comparación de las bibliotecas de interfaz gráfica estudiadas.**

<sup>9</sup> Capacidad del producto de software de ser modificado. Las modificaciones pueden incluir las correcciones, mejoras o adaptaciones del software a cambios en el ambiente [ISO/IEC 9126].

Como conclusión del análisis, se dedujo que:

En el indicador multiplataforma se evidencia que las bibliotecas Swing, SwingX y la Biblioteca de Componentes de Interfaz Gráfica de Usuario (BCGUI) de XEGFORT presentan esta propiedad, permitiéndoles el funcionamiento en los diferentes sistemas operativos. Por el contrario AWT no la presenta, teniendo la necesidad de componentes con diferentes configuraciones en las diversas plataformas. Los componentes de las bibliotecas Swing y SwingX presentan sincronización entre el modelo y el delegado, permitiéndoles la vista actualizada con cada cambio que se realiza en la información. No siendo así en AWT y en la BCGUI de XEGFORT.

Las bibliotecas AWT, Swing y SwingX presentan complejidad de las interfaces, causando que los componentes no cumplan con las facilidades para gestionar la información en el marco de trabajo de acuerdo al negocio. Permitiendo a la BCGUI de XEGFORT mejor gestión de la información en las interfaces gráficas. Por otra parte AWT no cumple con la propiedad de optimización del código, causando la necesidad de invertir tiempo en implementaciones adicionales y código repetitivo en el desarrollo de las interfaces gráficas.

El tiempo empleado en implementaciones adicionales es un indicador muy importante a cumplir, permitiéndole disminuir el tiempo de desarrollo de las aplicaciones que utilizan el marco de trabajo XEGFORT. Las bibliotecas estudiadas anteriormente tienen esta necesidad, debido a que emplean tiempo adicional en funcionalidades que le faciliten la gestión de la información de las aplicaciones que utilicen XEGFORT.

Se puede apreciar de forma general que las bibliotecas analizadas presentan deficiencias en todos los indicadores, ocasionando que el desarrollo de una biblioteca de componentes de interfaz gráfica basada en alguna de las estudiadas, pueda traer resultados insatisfactorios. Provocando la necesidad de realizar implementaciones adicionales y a la vez extendiendo el tiempo de desarrollo de las aplicaciones que utilizan XEGFORT.

El resultado del análisis de los indicadores permite determinar que es necesario desarrollar una biblioteca de componentes gráfica para aplicaciones de escritorio desarrolladas en Java, que permita disminuir el tiempo de desarrollo de las aplicaciones que utilizan el marco de trabajo XEGFORT. Permitiendo además, proporcionar solución al problema planteado y corregir las deficiencias encontradas en la actual biblioteca de componentes de XEGFORT, utilizando ventajas de nuevas tecnologías. Para dar solución a esta biblioteca es necesario el uso de una metodología que guíe el proceso de desarrollo del software.

## 1.2 Metodología para el desarrollo del software

Una metodología de desarrollo de software es un conjunto de pasos a seguir que guían la realización y documentación de un software. Estos pasos son la base para que todo proyecto se realice de forma correcta, entendible y con la mayor calidad posible (Sommerville, 2005).

Al no existir una metodología de software universal, y como toda metodología debe ser adaptada a las características de cada proyecto. La UCI toma la iniciativa de hacer una variación de la metodología *Agile Unified Process* (AUP por sus siglas en inglés), de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la entidad. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas (Mejora, 2013).

Esta metodología es una versión simplificada de *Unified Rational Process* (RUP por sus siglas en inglés). Variación AUP-UCI describe de una manera simple y fácil, la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. La selección de la metodología está dada debido a que el equipo de desarrollo presenta sólo dos integrantes y la planificación del cronograma de ejecución establece poco tiempo para darle respuesta al problema en cuestión. Además, por ser un proceso iterativo e incremental, posibilitando descomponer proyectos grandes en pequeños, permitiendo la detección temprana de riesgos. Ofrece la posibilidad de definir solamente los entregables que los desarrolladores consideren necesarios. Permite adaptarse a los cambios de los requisitos, así como detectar y gestionar los riesgos durante todo el ciclo de vida del proyecto (Mejora, 2013).

Otro elemento que influye en la selección de Variación AUP-UCI es que el cliente no forma parte del equipo de desarrollo y cuenta con poco tiempo para interactuar con el mismo. Además, proporciona una forma controlada de trabajar, guiando el orden de las actividades, tareas de los desarrolladores individuales y del equipo como un todo, siendo una metodología adaptable al contexto y necesidades (Mejora, 2013).

La metodología Variación AUP-UCI propone tres fases (Inicio, Ejecución, Cierre) y ocho disciplinas, de las que se utilizan siete para proporcionar solución a este trabajo (Modelado de negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas Internas, Pruebas de Liberación y Pruebas de Aceptación). Estas disciplinas guían el software desde su concepción hasta su entrega al usuario final (Mejora, 2013). Para constituir un modelo de cómo estará estructurado la biblioteca de interfaz gráfica y guiar el proceso de desarrollo de software, será necesario el uso de una arquitectura de software.

### 1.3 Arquitecturas de software

La arquitectura de software de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen los componentes del software, las propiedades visibles externamente de esos componentes y las relaciones entre ellos (Pressman, 2007). En este epígrafe se aborda las características de las arquitecturas basadas en componentes y Modelo-Vista-Controlador (MVC). Por ser las arquitecturas en las que se basa la biblioteca que incorpora el marco de trabajo XEGFORT y la biblioteca de interfaz gráfica Swing respectivamente.

#### Basada en componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales, los que se comunican a través de interfaces bien definidas que contienen métodos, eventos y propiedades (Peláez, 2009).

Los sistemas de software basados en componentes se fundamentan en principios definidos por una ingeniería de software específica, por lo que los componentes se constituyen como las unidades de modelado, diseño e implementación. Los componentes soportan algún régimen de introspección<sup>10</sup>, de modo que su funcionalidad y propiedades pueden ser descubiertas y utilizadas en tiempo de ejecución (Zaldivar, 2012).

Algunas de las ventajas que presenta esta arquitectura son:

- ✓ Las pruebas pueden ser ejecutadas probando cada uno de los componentes antes de que se pruebe todo el conjunto de componentes ensamblados.
- ✓ La calidad de una aplicación basada en componentes mejora con el tiempo, dado que un componente puede ser construido y mejorado seguidamente por un experto u organización.
- ✓ Permite una mayor reutilización del software.
- ✓ Simplifica el mantenimiento del sistema ya que cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

#### Modelo-Vista-Controlador

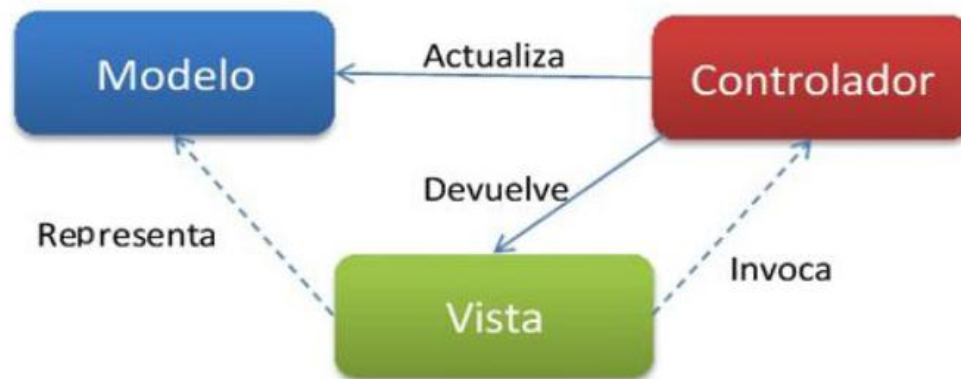
La arquitectura MVC tiene como objetivo principal separar el código responsable de la representación de los datos del código encargado de almacenarlos (de la Torre LLorente, 2010). Para lograrlo, la arquitectura divide la capa de presentación en tres tipos de objetos básicos; que son: modelos, vistas

---

<sup>10</sup> Observación y examen que una persona hace de sus propias ideas, pensamientos y sentimientos.



y controladores. La utilidad de esta reside en que describe los flujos de comunicación entre los tres tipos de objetos, como muestra la (Imagen 1):



**Imagen 1. Arquitectura de software Modelo-Vista-Controlador. Fuente:** (Frías, 2012)

En un escenario clásico, la vista constituye la representación visual de la GUI y el modelo representa la estructura lógica de la GUI, independientemente de su representación visual. 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 qué vista mostrar. Esta arquitectura permitirá reducir la complejidad en el diseño arquitectónico e incrementar la flexibilidad y mantenimiento del código (de la Torre LLorente, 2010).

Para el diseño arquitectónico del sistema, se hace uso del patrón arquitectónico de software Modelo-Vista-Controlador. Este patrón de software permite implementar cada uno de los componentes por separado, donde la vista y el controlador dependen de las interacciones del modelo. La conexión entre el modelo y sus vistas es dinámica, realizando el intercambio de información en tiempo de ejecución. La forma en que el patrón se estructura simplifica la comprensión y la organización del desarrollo del sistema, reduciendo las dependencias (de la Torre LLorente, 2010).

El patrón MVC fue diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales están dadas por el hecho de que, el modelo, las vistas y los controladores se tratan como entidades separadas; lo que hace que cualquier cambio producido en el modelo se refleje automáticamente en cada una de las vistas. Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas (Fernández Romero, y otros, 2012).

Este modelo de arquitectura presenta las siguientes ventajas:

- ✓ Separación clara entre los componentes de un programa; lo cual permite su implementación por separado.
- ✓ Interfaz de Programación de Aplicaciones (API) bien definida; cualquiera que use las API, podrá reemplazar el modelo, la vista o el controlador sin aparente dificultad.
- ✓ Conexión dinámica entre el modelo y sus vistas; se produce en tiempo de ejecución, no en tiempo de compilación.

Para el desarrollo de la investigación una vez seleccionada la arquitectura de software se utilizan patrones de diseño que brinden soluciones para problemas típicos y recurrentes que se pueden encontrar al desarrollar una aplicación.

### **1.4 Patrones de Diseño**

Los patrones son una descripción del problema y la esencia de su solución, de forma que se pueda reutilizar en diferentes situaciones. El patrón no es una especificación detallada, es una solución adecuada a un problema común (Sommerville, 2005). En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares (Gamma, y otros, 2005).

#### **Patrones generales de software para asignación de responsabilidades (GRASP)**

Los patrones GRASP describen un conjunto de principios para la asignación de responsabilidades (Diseño, 2014). Entre los más significativos se encuentran: experto, creador, bajo acoplamiento, alta cohesión y controlador.

- ✓ Experto: consiste en asignar una responsabilidad al experto en información, o sea, a la clase que cuenta con la información necesaria para cumplir la responsabilidad, garantizando el encapsulamiento de la información y facilitando el bajo acoplamiento en las aplicaciones.
- ✓ Creador: este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, brinda soporte al bajo acoplamiento.
- ✓ Bajo acoplamiento: se encarga de mantener las clases lo menos ligadas entre sí posible. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

- ✓ Alta cohesión: se basa en que la información que almacena una clase sea coherente y esté en la medida de lo posible relacionada con la clase. Con el uso de este patrón se logra un incremento de la claridad y comprensión, lo que simplifica el mantenimiento.
- ✓ Controlador: define que clase debe responder a determinados eventos. Una clase controladora debe ser la encargada de manejar los eventos dentro de la funcionalidad, así la aplicación del patrón conlleva a separar la lógica de negocios de la capa de presentación, al aplicar estos principios, el controlador no realiza las actividades mencionadas sino que las delega en otras clases.

### **Patrones grupo de los cuatro (GoF)**

Según su propósito los patrones GoF se clasifican en creacionales, estructurales y de comportamiento. El marco de trabajo de XEGFORT utiliza estos patrones, los cuales le permiten brindar solución a problemas recurrentes que se presentan (Holzner, 2006). Este marco de trabajo implementa funcionalidades que facilitan el uso del sistema.

- ✓ Creacionales: tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es 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.

La utilización de patrones de diseño es un aspecto que denota la existencia y necesidad de la reutilización. Debido que identifican aspectos claves de la estructura de un diseño que puede ser aplicado en diferentes situaciones. La reutilización es un principio clave con que cumple el estándar Especificación de Java *Beans*, seguidamente se muestra la descripción.

### **1.5 Especificación de Java Beans**

El lenguaje de programación Java ha utilizado la arquitectura de componentes Java *Beans* para representar la propiedad de un objeto. Este modelo de componentes es ampliamente entendido por desarrolladores de aplicaciones Java y herramientas de desarrollo por igual (Oracle Corporation, 2013).

El objetivo de la especificación de Java *Beans* es definir un modelo de componentes de software para Java. Esta especificación denomina sus componentes Java *Beans* los cuales se conciben cómo

componentes de software reutilizables que se pueden manipular visualmente en una herramienta de creación. Los componentes Java *Beans* poseen propiedades que pueden afectar su apariencia y/o comportamiento. Las propiedades son accedidas y modificadas a través de métodos de acceso (*Getters* y *Setters*) y se clasifican en propiedades indexadas, atadas o limitadas (Oracle Corporation, 2013).

Las tres características más importantes de Java *Beans* son el conjunto de propiedades que expone, el conjunto de métodos que permite que otros componentes llamen y el conjunto de eventos que lanza. Las propiedades pueden estar expuestas en entornos de guiones como si fueran campos de objetos. Se puede acceder mediante la programación por otros componentes que llaman sus métodos mediante los *get* y *set*. Como parte del proceso de personalización de un componente sus propiedades se pueden presentar en una hoja de propiedades para que un usuario pueda editarlas. Por lo general las propiedades de Java *Beans* serán persistentes, por lo que su estado se almacenará lejos como parte del estado persistente del *Bean* (JavaFX, 2013).

Los eventos proporcionan a un componente una forma de notificar a los demás que algo interesante ha sucedido. Son una de las características principales de la arquitectura Java *Beans*. Estos eventos proporcionan un mecanismo conveniente que permiten que los componentes sean conectados en un constructor de aplicaciones. Permitiendo a algunos componentes que actúen como fuentes para las notificaciones de eventos, que puede ser capturado y procesado por cualquiera de los entornos de secuencias de comandos o por otros componentes (Hamilton, December 1997).

Los métodos que presenta Java *Beans* son métodos normales de Java que se pueden llamar desde otros componentes. Por defecto, se supone que todos los métodos públicos de Java *Beans* deben ser expuestos como métodos externos dentro del entorno de componentes, para el acceso de otros componentes (Hamilton, December 1997).

El análisis de la Especificación de Java *Beans*, permite conocer las principales características y propiedades para definir un modelo de componentes de software, para los componentes de la biblioteca de interfaz gráfica en Java. Como una extensión de la Especificación de Java *Beans* se encuentra la Especificación de JavaFX *Beans*.

## 1.6 Especificación de JavaFX Beans

Java ha utilizado las API de Java *Beans* para representar una propiedad de un objeto. La propiedad de Java *Beans* está representada por un par de métodos *get* y *set*. Los cambios de propiedad se propagan a un cambio de propiedad a los oyentes a través de la activación de eventos de cambio en el código *set*. JavaFX 2.0 introduce la especificación JavaFX *Beans* que añade propiedades y apoyan a objetos Java a través de la ayuda de las clases de las propiedades de JavaFX 2.0 (Oracle Corporation, 2013).

Al igual que sus homólogos de Java *Beans*, las propiedades de JavaFX *Beans* son especificadas por un conjunto de métodos en una clase Java. Para definir una propiedad JavaFX *Beans* en una clase Java, proporcionan tres métodos: el *get*, el *set*, y el *get property*. Ejemplo de esto, para una propiedad denominada altura de tipo *double*, los tres métodos son:

- ✓ *public final double getHeight();*
- ✓ *public final void setHeight(double h);*
- ✓ *public DoubleProperty heightProperty();*

Los nombres de los métodos *get* y *set* siguen la convención Java *Beans*. Se obtienen concatenando *get* y *set* con el nombre de la propiedad con la primera letra en mayúscula. Para propiedades de tipo *boolean*, el nombre *get* también pueden comenzar con "is". El nombre de la propiedad *get* es obtenido mediante la concatenación del nombre de la propiedad "Property". Para definir la lectura de la propiedad JavaFX *Beans*, puede eliminar el método *set* o cambiarlo a un método privado y cambiar el tipo de retorno del *get* de la propiedad para hacer Sólo Propiedad de Lectura del inglés (*Read Only Property*) (Oracle Corporation, 2013).

Las propiedades de JavaFX se utilizan a menudo en combinación con la unión, un poderoso mecanismo para expresar relaciones directas entre las variables. Cuando los objetos participan en las consolidaciones, los cambios realizados en un objeto se reflejarán automáticamente en otro objeto. Esto puede ser útil en una variedad de aplicaciones, por ejemplo, en componentes como el menú desplegable. El cual debe adaptar su tamaño a las dimensiones de sus padres en la escena en que se encuentra, redimensionar su tamaño para plegarse o desplegarse según la acción. También puede ser utilizado en las propiedades del componente para formatear textos de acuerdo al tipo de expresión regular por la que debe regirse. O bien, la unión podría ser utilizada en una interfaz gráfica de usuario que mantiene automáticamente su visualización sincronizada con los datos subyacentes de la aplicación (Oracle Corporation, 2013).

Los enlaces se ensamblan a partir de una o más fuentes, conocidas como dependencias. Una unión observa su lista de dependencias para los cambios y luego se actualiza automáticamente después de que se ha detectado un cambio (Oracle Corporation, 2013). Una vez conocido el objetivo de estas especificaciones y su importancia en la implementación de los componentes GUI, está presente la plataforma de desarrollo JDK desde versiones tempranas. Por esta razón es necesario el conocimiento de las API y de plataformas que provee.

## 1.7 Tecnologías

La solución se desarrolla sobre el lenguaje de programación JavaFX, el cual es la evolución de la plataforma del cliente Java, diseñada para permitir a los desarrolladores crear y desplegar aplicaciones

que se comportan de forma consistente en múltiples plataformas. Construido sobre la tecnología Java, la plataforma JavaFX proporciona un conjunto de API de gráficos y contenidos multimedia de alto rendimiento con aceleración por hardware y los motores de los medios de comunicación que simplifican el desarrollo de aplicaciones. Utilizando la plataforma JavaSE, el kit de herramientas JDK 8, el conjunto de API de gráficos y el lenguaje de marcado FXML basado en XML, se dispone de programas y librerías para la creación de interfaces visuales que agilicen la programación y permitan crear fácilmente interfaces de usuario complejas.

### **JDK 8**

El Kit de Herramientas Java (JDK) es un conjunto de herramientas útiles para desarrollar y probar programas implementados en el lenguaje de programación Java (Oracle Corporation, 2013).

Incluye una gran actualización para el modelo de programación Java, así como una evolución coordinada de la JVM<sup>11</sup>, el lenguaje Java y las bibliotecas. Java 8 incluye funciones para la productividad, facilidad de uso, mejora de la programación políglota<sup>12</sup>, seguridad, rendimiento mejorado y la integración con la tecnología JavaFX, propiciando las herramientas, librerías y programas necesarios para el desarrollo de aplicaciones (Oracle Corporation, 2013). JDK 8 presenta soporte para las plataformas JavaEE, JavaSE y JavaME<sup>13</sup> para el desarrollo de las aplicaciones. Para dar solución a este trabajo, se hace uso de la plataforma JavaSE.

### **Plataforma JavaSE**

La plataforma JavaSE provee herramientas de desarrollo así como programas y librerías para la creación de programas en Java. Es un lenguaje de programación orientado a objetos. Es independiente de la plataforma, esto significa que se puede ejecutar en cualquier sistema operativo que contenga JVM. Java toma mucho 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). Para el desarrollo en el lenguaje Java, se ofrece un grupo de interfaces e implementaciones de referencia orientadas a la solución de problemas específicos como el trabajo con ficheros XML<sup>14</sup>, persistencia y transacciones. Todas ellas estructuradas como API, concepto que se presenta a continuación.

---

<sup>11</sup> Máquina Virtual de Java.

<sup>12</sup> Texto escrito en varias lenguas.

<sup>13</sup> Edición Micro de Java.

<sup>14</sup> Lenguaje de marcado extensible.

## API

La Interfaz de programación de aplicaciones, constituye un conjunto de comandos, funciones y protocolos que los programadores pueden utilizar en la construcción de software para un sistema operativo específico. Las API permiten a los programadores utilizar funciones predefinidas para interactuar con el sistema operativo, en lugar de escribir desde cero. Son usadas generalmente en las bibliotecas gráficas (Christensson, 2015). Para dar solución a la biblioteca de componentes gráficos, se utiliza JavaFX como lenguaje de programación.

## JavaFX

JavaFX es una familia de productos y tecnologías de *Sun Microsystems*<sup>15</sup>, adquirida por *Oracle Corporation*<sup>16</sup>. Constituye una plataforma que contiene un ambiente de ejecución multiplataforma. Soporta las librerías de Java para aprovechar sus potencialidades. Cuenta con su propio lenguaje declarativo al igual que con un conjunto de herramientas de diseño y desarrollo, lo cual permite crear contenido expresivo, creativo, dinámico y funcional. JavaFX contiene un conjunto de tecnologías y herramientas para el diseño, desarrollo y despliegue de aplicaciones de contenido expresivo, para ser visualizado en equipos con pantalla (Oracle Corporation, 2013).

Las aplicaciones desarrolladas empleando esta tecnología pueden utilizar las bibliotecas API de Java para acceder a las capacidades nativas de los sistemas y conectarse a las aplicaciones de middleware<sup>17</sup> basadas en servidor. Presenta características como: soporte para aplicaciones AWT y Swing; empaquetado de aplicaciones autónomas con instaladores específicos de la plataforma; JavaFX Scene Builder como herramienta de diseño de interfaz de usuario visual y FXML (ver en el siguiente acápite) como nuevo lenguaje de marcado basado en XML para la definición de interfaces de usuario (JavaFX, 2013).

Como propiedad especial para las interfaces visuales JavaFX utiliza las Hojas de Estilo en Cascada (CSS por sus siglas en inglés) de apariencia independiente y estilo de la aplicación para que los desarrolladores puedan concentrarse en la codificación. Los diseñadores gráficos pueden personalizar fácilmente la apariencia y el estilo de la aplicación a través de CSS. En caso de existir un diseño de fondo, o se desea separar la interfaz de usuario y la lógica de fondo, entonces JavaFX permite

---

<sup>15</sup> Empresa informática que se dedicaba a vender estaciones de trabajo, servidores, componentes informáticos, software, sistemas operativos y servicios informáticos.

<sup>16</sup> Una de las mayores compañías de software del mundo.

<sup>17</sup> Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos.

desarrollar los aspectos de la presentación de la interfaz de usuario en el lenguaje FXML y utilizar el código de Java para la lógica de la aplicación (JavaFX, 2013).

JavaFX para su ejecución propone una arquitectura, en la que describe el motor que ejecuta el código y cómo se interconectan cada uno de los componentes que presenta su arquitectura. Esta tecnología se compone de subcomponentes que incluyen el nuevo motor de gráficos de alto rendimiento, llamado Prisma (del inglés *Prism*). También utiliza el nuevo sistema de ventanas pequeño y eficiente llamado Cristal (del inglés *Glass*), que es un motor de medios de comunicación y un motor para aplicaciones web (Oracle Corporation, 2013). A pesar de que estos componentes no están expuestos públicamente, sus descripciones pueden ayudar a entender cómo se ejecuta una aplicación JavaFX (Ver Anexo 2). Para definir las interfaces de usuario de una aplicación se hace uso del lenguaje de codificación FXML.

## FXML

Por su traducción *FX Markup Language*, es un lenguaje de codificación de secuencias de comandos de marcado basado en XML para la construcción de gráficos de objetos en JavaFX. Proporciona una alternativa conveniente para la construcción de gráficos en el código de procedimiento. Es ideal para la definición de la interfaz de usuario de una aplicación JavaFX, debido a que la estructura jerárquica de un documento XML se asemeja mucho a la estructura del escenario gráfico (FXML, 2014).

FXML utiliza tipos de coacción<sup>18</sup> para convertir los valores de propiedad en el tipo adecuado según sea necesario. Se requiere tipos de coacción porque los únicos tipos de datos compatibles con XML son elementos, texto y atributos cuyos valores son también el texto. Sin embargo, Java soporta un número de diferentes tipos de datos, incluyendo los tipos de valor primitivo incorporados, así como los tipos de referencia extensibles (FXML, 2014).

El cargador FXML utiliza el método *coerce()* de *BeanAdapter*<sup>19</sup> para realizar las conversiones de tipos requeridos. Este método es capaz de realizar conversiones básicas de tipo primitivo como *String* a *boolean* o *int* a *double*; y también convertir *String* a *class* o *String* a *Enum*. Las conversiones adicionales pueden implementarse mediante la definición de un método estático *valueOf()* sobre el tipo de destino (FXML, 2014). Durante el desarrollo de la aplicación se utilizan herramientas para apoyar el desarrollo de la propuesta de solución.

## 1.8 Herramientas

Las herramientas de Ingeniería de Software Asistida por Computadoras (CASE), permiten aplicar la metodología de análisis y diseño orientado a objeto y abstraerse del código fuente. Estos programas

---

<sup>18</sup> Convierte valores de propiedad en el tipo de datos adecuados.

<sup>19</sup> Clase encargada de convertir múltiples propiedades de Java *Beans* en modelos de valor.



de ayuda proporcionan asistencia a los analistas, diseñadores y desarrolladores durante todo el ciclo de vida de desarrollo de un software (Sommerville, 2005). Entre los principales objetivos de este tipo de herramienta se encuentran:

- ✓ Permite el desarrollo de todas las fases del software con una misma herramienta.
- ✓ Facilita el uso de las distintas metodologías, lo que permite generar artefactos por cada una de las disciplinas que esta propone.

Para realizar el modelado del sistema se define el Lenguaje de Modelado Unificado (UML).

### **UML 2.0**

UML está definido como un lenguaje para visualizar y documentar los artefactos de un sistema de software orientado a objetos. Permite transformar en un lenguaje estándar los componentes del proceso de desarrollo de aplicaciones. Su objetivo principal, es entregar un material de apoyo que permita definir diagramas propios, como también entender el modelado de diagramas existentes. Es un lenguaje estándar de modelado de software, para la visualización, especificación, construcción y documentación de los artefactos de sistemas en los que el software juega un papel importante. Permite a los desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados (Jacobson, 2000). Para la solución del trabajo se realizó el diagrama de modelo conceptual, diagramas de clases y diagrama de componentes.

Para facilitar el uso de este lenguaje de modelado se utiliza la herramienta Visual Paradigm. La selección de esta herramienta viene dada por las especificaciones de la metodología de desarrollo, donde es propuesta para el trabajo del Análisis y Diseño. Además fue seleccionada por las facilidades de uso y la amplia gama de utilidades que brinda. A continuación se realiza una descripción de la herramienta.

### **Visual Paradigm 8.0 para UML**

Es una herramienta con entorno de creación de diagramas. Utiliza UML 2.0 como lenguaje de modelado, con soporte multiplataforma. Soporta un conjunto de lenguajes de programación para la generación de código e ingeniería inversa. Permite representar diferentes tipos de diagramas así como diagramas de clases, modelo conceptual, componentes, paquetes, entre otros. Presenta apoyo adicional en cuanto a generación de artefactos automáticamente. Soporta el ciclo de vida completo del desarrollo de software sin necesidad de utilizar herramientas externas (Visual Paradigm, 2013).

Luego del uso de esta herramienta CASE para la representación visual de los principales componentes del desarrollo del software, se dio paso a la caracterización de la herramienta NetBeans 8.0.

## **Entorno de desarrollo integrado**

Un entorno de desarrollo integrado (IDE), es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos (Oracle Corporation, 2013).

Para la implementación de la solución de software se utiliza NetBeans 8.0 como IDE de desarrollo, con un conjunto de complementos que amplían sus funcionalidades como ambiente de desarrollo integrado de aplicaciones de escritorio. Además, por ser la versión que soporta la JDK 8.

### **NetBeans 8.0**

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Es un producto gratuito sin restricciones de uso. Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos, debido a que estos pueden ser desarrollados independientemente. Las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software (Oracle Corporation, 2013).

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están: administración de las interfaces de usuario (ejemplo: menús y barras de herramientas), administración de las configuraciones del usuario y administración del almacenamiento (Oracle Corporation, 2013). Este IDE de desarrollo se utiliza además, por su integración con la herramienta de diseño JavaFX *Scene Builder*.

### **JavaFX Scene Builder 2.0**

*Scene Builder* 2.0 es una herramienta de diseño para la creación de las interfaces gráficas. Es un entorno de diseño visual que permite diseñar rápidamente interfaces de usuario para aplicaciones JavaFX, sin necesidad de escribir ningún código fuente. Permite arrastrar y soltar los componentes en la interfaz gráfica, facilitando su manipulación a la hora de posicionar los elementos en JavaFX *Scene Builder*. A medida que se construye el diseño de interfaz, se genera automáticamente el código FXML para el diseño (Oracle Corporation, 2013).

JavaFX *Scene Builder* proporciona una interfaz simple e intuitiva que puede ayudar incluso a los no programadores a crear rápidamente prototipos de aplicaciones interactivas, que conectan componentes de interfaz gráfica de usuarios a la lógica de la aplicación. Los diseñadores y

desarrolladores de Java pueden crear prototipos de interfaz gráfica de usuarios para el diseño de la aplicación cliente y el desarrollo de la lógica de la aplicación por separado. También pueden obtener una vista previa del diseño de interfaz gráfica de usuario y definir su apariencia con las hojas de estilo (Oracle Corporation, 2013).

JavaFX *Scene Builder* incluye las siguientes características:

- ✓ Presenta una interfaz *What You See Is What You Get* (WYSIWYG por sus siglas en inglés), capaz de arrastrar y soltar los componentes, permitiendo crear rápidamente un diseño de interfaz gráfica de usuario sin la necesidad de escribir código fuente.
- ✓ Posee generación automática de código FXML y se produce a medida que construye y modifica su diseño GUI.
- ✓ Posee funciones de edición y vista previa en tiempo real que le permiten visualizar rápidamente los cambios de diseño de interfaz gráfica de usuario, que se realicen sin la necesidad de compilar. Esta característica ayuda a minimizar el tiempo de desarrollo de la aplicación.
- ✓ Permite control completo de acceso a la interfaz gráfica de usuario JavaFX que proporciona la biblioteca.
- ✓ JavaFX posibilita añadir componentes de interfaz gráfica personalizados por el usuario a la biblioteca. La biblioteca de componentes de GUI disponible, se puede ampliar mediante la importación de componentes de GUI personalizados desde archivos Java (jar) de terceros, archivos fxml, o agregarlos desde la jerarquía o paneles de contenido.
- ✓ Presenta soporte para CSS el cual permite una flexible gestión de la apariencia de la interfaz de usuario de la aplicación (Oracle Corporation, 2013).

En el empaquetado de los componentes al diseñar la interfaz de usuario, el *Scene Builder* crea una clase FXML que puede ser exportada a un IDE, de modo que los desarrolladores puedan añadir la lógica empresarial.

Luego de seleccionadas las herramientas para el desarrollo de la biblioteca de componentes, se dio paso a la selección de técnicas y métricas aplicables a la validación de los requisitos y el diseño.

### **1.9 Técnicas y Métricas**

Las técnicas y métricas para el software proporcionan una manera sistemática de valorar la calidad basándose en un conjunto de reglas. También proporcionan al ingeniero del software descubrir y corregir problemas potenciales antes de que se conviertan en defectos graves (Software, 2003). En el caso de la presente investigación se utilizan para la validación de los requisitos y el diseño.

## Técnicas para los requisitos

**Construcción de prototipos:** son la versión inicial de un sistema de software que se utiliza para demostrar los conceptos, probar las opciones de diseño y entender mejor el problema y su solución, permiten revelar errores u omisiones en los requisitos propuestos (Prototipos, 2015).

**Revisión:** es una técnica formal, que involucra tanto a los clientes cómo a los desarrolladores, su objetivo es detectar manualmente anomalías u omisiones en cualquier producto de software a partir de la revisión del documento de requisitos (Sommerville, 2005).

## Métrica para los requisitos

**Especificidad de Requisitos:** la comprensibilidad de los requisitos depende en gran medida de la ausencia de ambigüedades en su especificación, facilitando los procesos de captura y procesamiento de requisitos. El objetivo de esta métrica es cuantificar la especificidad o falta de ambigüedad en la definición de los requisitos (Pressman, 2005).

Para calcular esta métrica deben contarse los requisitos que tuvieron igual interpretación por los revisores y compararlos con el total de requisitos definidos. La especificidad de los requisitos se calcula como:

$$Q_1 = \frac{n_{ui}}{n_r}$$

### Fórmula para calcular la especificidad de requisitos

Donde:

- ✓  $Q_1$ : Grado de especificidad de los requisitos.
- ✓  $n_{ui}$ : Número de requisitos para los que los revisores tuvieron interpretaciones idénticas.
- ✓  $n_r$ : Cantidad de requisitos en una especificación y comprende la suma de los requisitos funcionales y no funcionales.

## Métrica para el diseño

Permiten medir de forma cuantitativa la calidad de los atributos internos del software. Esto permite al ingeniero evaluar la calidad durante el desarrollo de la aplicación (IngSoftwareII-CUFM, 2012).

## Tamaño operacional de clases (TOC)

Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- ✓ Responsabilidad: un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.

- ✓ Complejidad de implementación: un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- ✓ Reutilización: un aumento del TOC implica una disminución del grado de reutilización de la clase.

### **Relaciones entre clases (RC)**

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- ✓ Acoplamiento: un aumento del RC implica un aumento del Acoplamiento de la clase.
- ✓ Complejidad de mantenimiento: un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- ✓ Reutilización: un aumento del RC implica una disminución en el grado de reutilización de la clase.
- ✓ Cantidad de pruebas: un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Para validar el diseño se escoge la métrica Tamaño operacional de clases (TOC). Esta métrica básica, está inspirada en el estudio de la calidad del diseño orientado a objeto, teniendo en cuenta que brinda un esquema sencillo de implementación y que a la vez cubre los principales atributos de calidad de software (Diseño, 2015). La métrica Relaciones entre clases, no se utiliza, debido a que no se ajusta a la estructura de clases que presentan los componentes de la biblioteca, puesto que funcionan de manera individual y no se relacionan entre ellos.

Luego de definidas las técnicas y métricas para la validación de las funcionalidades y el diseño, se establecieron las pruebas de software para validar los resultados de la implementación.

### **1.10 Pruebas**

Uno de los elementos principales para certificar la calidad de una aplicación informática lo constituye el resultado de las pruebas que se practican. Un control y una gestión de calidad implementados de forma adecuada, especialmente durante el proceso de desarrollo del software, aumentan la calidad del sistema final, reducen los costos de avance y acortan el tiempo necesario para el desarrollo. Para determinar el nivel de calidad se deben efectuar medidas o pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones principales del sistema (Presman, 2005).

#### **Objetivo de las pruebas:**

- ✓ Validar y probar los requisitos que debe cumplir el software.

- ✓ Verificar que los requisitos fueron implementados correctamente.
- ✓ Encontrar y documentar los defectos que puedan afectar la calidad del software.
- ✓ Verificar que el software trabaje como fue diseñado.

Las siguientes disciplinas de pruebas de software que se describen a continuación son propuestas por la metodología de desarrollo utilizada Variación AUP-UCI:

**Pruebas internas:** son realizadas por sus propios desarrolladores, verificando el resultado de la implementación, probando cada construcción según sea necesario, así como las versiones finales a ser liberadas. Los artefactos necesarios para la realización de estas pruebas son los casos de pruebas (Mejora, 2013). Las pruebas definidas para desarrollar a nivel de unidad son las pruebas funcionales utilizando los métodos de prueba de caja blanca y caja negra.

Los métodos de prueba del software tienen el objetivo de diseñar pruebas que descubran diferentes tipos de errores con menor tiempo y esfuerzo. Ayudan a definir conjuntos de casos de prueba aplicando ciertos criterios. Los métodos de prueba que se especifican a continuación permiten probar cada una de las condiciones existentes en el programa, identificar claramente las entradas, salidas y estudiar las relaciones que existen entre ellas, permitiendo así maximizar la calidad de las pruebas y de este modo obtener un sistema más estable y confiable (Pressman, 2007).

### **Caja Blanca**

El método de prueba de caja blanca también suele ser llamada estructurales o prueba de caja de cristal. Con ella se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico. Realiza un seguimiento del código fuente según se va ejecutando los casos de prueba que ejecutan cada posible ruta del programa o módulo, considerándose una ruta como una combinación específica de condiciones manejadas por un programa (Pressman, 2007). Este tipo de prueba le permite al ingeniero de software obtener casos de pruebas que:

- ✓ Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada método.
- ✓ Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
- ✓ Ejecuten todos los bucles en sus límites operacionales.

Para efectuar el método de prueba de caja blanca se utiliza la técnica del camino básico. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la

definición de un conjunto de caminos básicos. Los casos de prueba obtenidos garantizan que durante la prueba se ejecute al menos una vez cada sentencia del programa.

### **Caja Negra**

El método de prueba de caja negra también denominado pruebas de comportamiento se concentran en los requisitos funcionales del software. Con ellas se pretende examinar el programa para verificar que cuente con las funcionalidades que debe tener y la forma de llevar a cabo las mismas, analizando siempre los resultados que devuelve y probando todas las entradas en sus valores válidos e inválidos. Las pruebas no se hacen en base al código, sino a la interfaz (Pressman, 2007).

Para efectuar el método de prueba de caja negra se utiliza la técnica partición equivalente. Esta divide el dominio de entrada de un programa en clases de datos de los que pueden derivarse casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. Se basa en una evaluación de las clases de equivalencia para una condición de entrada.

**Pruebas de Liberación:** son diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación (Mejora, 2013). El Centro de Gobierno Electrónico establece como entidad certificadora al Grupo de calidad de CEGEL, para realizar el método de prueba de caja negra utilizando la técnica partición equivalente, para las pruebas de liberación de la aplicación.

**Pruebas de aceptación:** son las únicas pruebas que son realizadas por los clientes. Consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario (Mejora, 2013).

### **Conclusiones Parciales**

- ✓ El análisis de las bibliotecas de interfaz gráficas estudiadas permitió obtener como resultado la necesidad de crear una biblioteca de componentes para aplicaciones de escritorio desarrolladas en Java que permita disminuir el tiempo de desarrollo de las interfaces visuales.
- ✓ El estudio de la metodología Variación AUP-UCI permitió guiar y profundizar en la investigación de las tareas vinculadas con el proceso de desarrollo de software.
- ✓ Se analizaron las arquitecturas de software adecuadas para el diseño de la estructura de la biblioteca de componentes y el uso de los patrones de diseño para obtener diferentes soluciones a los problemas de desarrollo.
- ✓ Se investigaron los modelos de componentes de Java y JavaFX para conocer el funcionamiento de sus estructuras, así como las propiedades, métodos y eventos para su reutilización en la implementación de la biblioteca de componentes.

# 2

## DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN

*Cuando estoy trabajando en un problema, nunca pienso en la belleza.  
Sólo pienso en cómo resolver el problema. Pero cuando he acabado,  
si la solución no es hermosa, sé que es un error.*

*Buckminster Fuller*

### Introducción

El presente capítulo se enfoca en describir los artefactos a realizar por cada una de las disciplinas que propone la metodología de desarrollo. Se describe el funcionamiento de la biblioteca de componentes dejando claro su objetivo principal, se realiza el diagrama de modelo conceptual donde se esclarecen los conceptos más relevantes y cómo se encuentran relacionados. Se definen los requisitos funcionales y no funcionales, además de sus descripciones. Se describe el diseño arquitectónico, además, del diagrama de clases y los patrones de diseño empleados. Se realiza una breve descripción de los componentes que integran la biblioteca de interfaz gráfica de usuario y se define el diagrama de componentes para apoyar la implementación.

### 2.1 Modelado de negocio

Esta disciplina está destinada a comprender el funcionamiento de la biblioteca de componentes. Se realiza el modelado del negocio mediante el diagrama de modelo conceptual generándose el artefacto Modelo Conceptual, donde quedarán establecidas las principales definiciones a tratar. El modelo conceptual sirvió como mapa explicativo para esclarecer los conceptos en el dominio del problema y los atributos pertenecientes a cada uno de los conceptos.

### Descripción de la Biblioteca de Componentes

La biblioteca de componentes que se desea implementar está basada en las clases, componentes y el conjunto de ventajas que brinda la herramienta JavaFX *Scene Builder* 2.0, haciendo uso de la JDK 8 que presenta soporte para la tecnología JavaFX. Esta biblioteca propone a las aplicaciones que hacen uso de XEGFORT agilizar el tiempo de desarrollo en las interfaces visuales. Brindándoles a los usuarios desarrolladores la ventaja de crear interfaces en el *Scene Builder* con la facilidad de arrastrar y soltar los componentes en la herramienta, fácil manipulación de sus propiedades y atributos, incluyendo el uso de CSS para proporcionar mejor calidad visual.

Una vez creada la escena que representa el formulario, se puede exportar una clase de extensión fxml la cual integra y describe cada uno de los componentes en su interior, con las especificaciones de sus propiedades, así como dimensiones, colores, CSS, entre otros. Mediante la herramienta IDE Netbeans, el desarrollador puede conformar los componentes utilizando las propiedades y clases de JavaFX y el



formulario fxml para exportar el componente en un fichero jar. Capaz de almacenar todos los eventos, operaciones y funcionalidades que presente el componente desarrollado para su posterior uso.

En el momento de haber sido importados los componentes de interfaz gráfica en la herramienta *Scene Builder*, el usuario desarrollador podrá utilizarlos de la misma forma que los originales que se encuentran integrados en la herramienta de diseño. Haciendo reutilización de las clases de JavaFX, la biblioteca cuenta con 12 componentes de interfaz visual, así como tablas, formateadores de texto, componentes para mostrar listados de elementos, carteles (banners) para mostrar en las aplicaciones, entre otros. Ver diagrama (Imagen 15).

Estos componentes permiten facilitar el trabajo a los desarrolladores, debido a que cuentan con propiedades tan avanzadas que el desarrollador sólo debe especificar el nombre del campo a mostrar en los componentes e insertarle un ObservableList con los elementos a mostrar. Además, de otras funcionalidades capaces de reducir la complejidad del uso de los componentes. Un ObservableList es una lista perteneciente a la colección de listas de JavaFX capaz de admitir diferentes tipos de datos para su uso, muy similar a las listas de Swing. La cual se utiliza en los componentes que requieren mostrar y manipular datos por los usuarios.

La biblioteca de componentes haciendo uso de JavaFX está conformada por componentes simples y compuestos. En el caso de los simples sólo reutilizan las propiedades de una clase específica de JavaFX, utilizando sus operaciones y métodos, y en algunos casos redefiniendo algunos métodos particulares. Los componentes compuestos extienden de una clase que permite encapsular varias de ellas permitiendo conformar todas las funciones del componente, y mostrando las propiedades tanto privadas como públicas para que puedan ser vistas desde otros componentes. A continuación se muestra la descripción y representación visual del modelo conceptual de la biblioteca de componentes para modelar la estructura de la propuesta de solución.

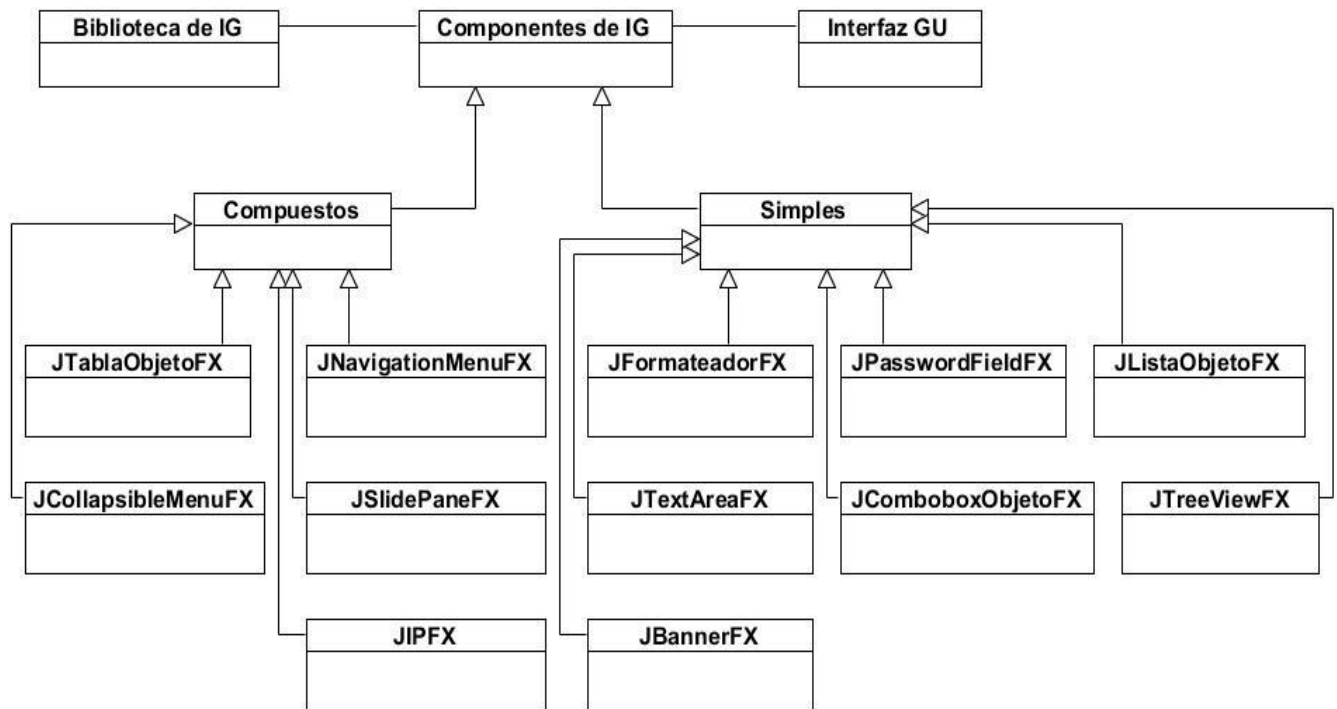
### Modelo conceptual

El modelo conceptual propuesto (Imagen 2) se realizó con el objetivo de identificar y explicar los conceptos significativos en la biblioteca de componentes, identificando los atributos y las relaciones existentes entre ellos.

- ✓ **Biblioteca de IG:** una biblioteca de interfaz gráfica es una colección o conjunto de componentes usados para desarrollar interfaces gráficas.
- ✓ **Componentes de IG:** los componentes de interfaz gráfica son elementos de un sistema de software que ofrecen un conjunto de servicios o funcionalidades.

- ✓ **Interfaz GU:** la interfaz gráfica de usuario está compuesta por un conjunto de componentes que hacen que la interfaz sea más amigable al usuario. Está diseñada para brindar al usuario final la manera ágil de encontrar y recordar el uso de las opciones que más le interesan.
- ✓ **Compuestos:** son los componentes que necesitan de otros componentes para su desarrollo.
- ✓ **Simples:** son los componentes que no requieren la composición de otros objetos o componentes gráficos.
- ✓ **JTablaObjetoFX, JNavigationMenuFX, JCollapsibleMenuFX, JIPFX, JSlidePaneFX:** son los componentes compuestos que integran la biblioteca de interfaz gráfica de usuario.
- ✓ **JFormateadorFX, JComboboxObjetoFX, JListaObjetoFX, JTextAreaFX, JTreeViewFX, JPasswordFieldFX, JBannerFX:** son los componentes simples que integran la biblioteca de interfaz gráfica de usuario.

A continuación se muestra el modelo conceptual de la biblioteca de componentes sin la representación de los atributos; para un mayor nivel de detalle (ver Anexo 3):



**Imagen 2. Diagrama de Modelo Conceptual.**

El modelado de negocio provee una vista estática de la estructura de la biblioteca de componentes, por lo que se define el modelo conceptual para su representación. Seguido de esta disciplina se da paso al levantamiento y captura de requisitos. A continuación se ofrece una descripción de las actividades en la disciplina requisitos.

## 2.2 Requisitos

Esta disciplina se enfocó en comprender la administración y gestión de los requisitos funcionales y no funcionales del producto. Se realizó el levantamiento de requisitos, tanto funcionales como no funcionales, generándose el artefacto Descripción de Requisitos Ágil para la biblioteca de componentes de interfaz de usuario, que constituye un elemento clave para el diseño y la posterior implementación de las funcionalidades.

### Técnica para la captura de requisitos funcionales

Una adecuada comprensión de los requisitos favorece el desarrollo de nuevos sistemas informáticos que cumplan con las necesidades y expectativas del cliente. Para realizar este procedimiento existen diversas técnicas que guían al analista en el proceso de redacción de un requisito funcional. Para la captura de los requisitos en el trabajo se utilizaron las siguientes técnicas:

**Entrevista:** es de gran utilidad para obtener información cualitativa, requiere seleccionar bien a los entrevistados para obtener la mayor cantidad de información en el menor tiempo posible. Es muy aceptada y permite acercarse al problema de una manera natural (Beyris Souly, et al., 2010). Esta técnica fue aplicada a los especialistas del centro CEGEL pertenecientes a los proyectos SIGFE, SIRECC y SIGESAP.

**Tormentas de ideas:** para la aplicación de esta técnica se realizaron varios encuentros entre los miembros del equipo de desarrollo donde ambas partes brindaban sus ideas en cuanto a la propuesta de solución. Esta técnica se puede utilizar para identificar un primer conjunto de requisitos en aquellos casos donde no están muy claras las necesidades que hay que cubrir (Beyris Souly, et al., 2010).

### Requisitos Funcionales

Los requisitos funcionales describen lo que el sistema debe hacer y cómo debe comportarse en situaciones específicas (Sommerville, 2005). Como resultado del uso de las técnicas de obtención de requisitos se identificaron 62 requisitos funcionales y 8 requisitos no funcionales. A continuación se presentan los requisitos funcionales del sistema.

#### Listado de los requisitos funcionales (RF):

Requisitos Funcionales	
No.	Requisitos
RF-1	Validar formato de una cadena de caracteres en JFormateadorFX
RF-2	Restringir entrada de caracteres en JFormateadorFX.
RF-3	Selección de expresiones regulares definidas en las propiedades del JFormateadorFX.
RF-4	Importar lista de elementos en JComboboxObjetoFX.
RF-5	Especificar atributo de los valores de la lista a mostrar en JComboboxObjetoFX.

RF-6	Mostrar elemento seleccionado en JComboboxObjetoFX.
RF-7	Obtener objeto de elemento seleccionado en JComboboxObjetoFX.
RF-8	Eliminar elemento seleccionado en JComboboxObjetoFX.
RF-9	Plegar menú en JCollapsibleMenuFX.
RF-10	Desplegar menú en JCollapsibleMenuFX.
RF-11	Mostrar funcionalidades de los flujos de trabajo en el menú en JCollapsibleMenuFX.
RF-12	Importar lista de elementos en JTablaObjetoFX.
RF-13	Mostrar valores de atributos especificados en JTablaObjetoFX.
RF-14	Mostrar nombre de columnas mediante atributos especificados en JTablaObjetoFX.
RF-15	Mostrar lista de elementos en JTablaObjetoFX.
RF-16	Seleccionar tupla en JTablaObjetoFX.
RF-17	Seleccionar tuplas múltiples en JTablaObjetoFX.
RF-18	Obtener objeto seleccionado en JTablaObjetoFX.
RF-19	Eliminar tupla seleccionada en JTablaObjetoFX.
RF-20	Eliminar selección múltiple de tuplas en JTablaObjetoFX.
RF-21	Paginar tabla en JTablaObjetoFX.
RF-22	Filtrar búsqueda de elementos en JTablaObjetoFX.
RF-23	Ocultar paginado en JTablaObjetoFX.
RF-24	Ocultar filtro de búsqueda en JTablaObjetoFX.
RF-25	Registrar dirección IP en JIPFX.
RF-26	Restringir entrada de números enteros no contenidos en el intervalo 0 a 255 en JIPFX.
RF-27	Obtener dirección IP insertada en JIPFX.
RF-28	Mostrar camino que sigue un usuario dentro de la aplicación en JNavigationMenuFX.
RF-29	Validar existencia única de ubicación en JNavigationMenuFX.
RF-30	Importar lista de elementos en JListaObjetoFX.
RF-31	Especificar atributo de los valores de la lista a mostrar en JListaObjetoFX.
RF-32	Mostrar lista vertical de elementos en JListaObjetoFX.
RF-33	Seleccionar elemento en JListaObjetoFX.
RF-34	Seleccionar selección múltiple de elementos en JListaObjetoFX.
RF-35	Obtener objeto de elemento seleccionado en JListaObjetoFX.
RF-36	Eliminar elemento seleccionado en JListaObjetoFX.
RF-37	Eliminar selección múltiple de elementos en JListaObjetoFX.
RF-38	Mostrar cartel (banner) para aplicaciones de escritorio en JBannerFX.
RF-39	Conservar aspecto original de la imagen en JBannerFX.
RF-40	Modificar tamaño de la imagen sin distorsión en JBannerFX.
RF-41	Ocultar caracteres de la contraseña en JPasswordFieldFX.
RF-42	Modificar carácter que oculta el texto plano de la contraseña en JPasswordFieldFX.
RF-43	Mostrar contraseña registrada en JPasswordFieldFX.
RF-44	Almacenar inserción de textos en JTextAreaFX.
RF-45	Mostrar texto insertado en JTextAreaFX.
RF-46	Ajustar líneas de texto en JTextAreaFX.
RF-47	Validar entrada de caracteres en tiempo de ejecución en JTextAreaFX.
RF-48	Cambiar estilo del formato de texto normal a texto en negrita en tiempo de ejecución en JTextAreaFX.
RF-49	Cambiar estilo del formato de texto negrita a texto normal en tiempo de ejecución en JTextAreaFX.
RF-50	Cambiar estilo del formato de texto normal a texto cursivo en tiempo de ejecución en JTextAreaFX.

RF-51	Cambiar estilo del formato de texto cursivo a texto normal en tiempo de ejecución en JTextAreaFX.
RF-52	Deslizar secuencia de paneles hacia delante en JSlidePaneFX.
RF-53	Deslizar secuencia de paneles hacia atrás en JSlidePaneFX.
RF-54	Gestionar botones de navegación de los flujos de paneles en JSlidePaneFX.
RF-55	Habilitar botones de navegación automáticamente en JSlidePaneFX.
RF-56	Deshabilitar botones de navegación automáticamente en JSlidePaneFX.
RF-57	Importar lista de elementos en JTreeViewFX.
RF-58	Especificar valor a mostrar de nodo raíz en JTreeViewFX.
RF-59	Especificar valor a mostrar de atributo de agrupación en JTreeViewFX.
RF-60	Mostrar objeto de elemento seleccionado en JTreeViewFX.
RF-61	Mostrar lista de elementos en forma de árbol (o por jerarquía) en JTreeViewFX.
RF-62	Eliminar elemento seleccionado en JTreeViewFX.

**Tabla 2. Listado de los requisitos funcionales.**

### Descripción de los Requisitos Funcionales

En el proceso de desarrollo de la biblioteca de componentes fue fundamental la descripción de los requisitos funcionales, estos facilitaron un entendimiento de los procesos a desarrollar, permitiendo comprender el problema en cuestión y facilitando la identificación de las funcionalidades que serán implementadas. La especificación de requisitos, es la base que permite verificar si se alcanzaron o no los objetivos establecidos en el proyecto.

En las (Tablas 4 y 5) se muestra un resumen de la descripción textual de los requisitos *Validar formato de una cadena de caracteres en JFormateadorFX* y *Mostrar elemento seleccionado en JComboboxObjetoFX*. En el documento *Descripción de Requisitos Ágil* propuesto por (Durán Bolaño, y otros, 2015), ubicado en la carpeta expediente de proyecto BCGUI, donde se muestran las descripciones de todos los requisitos funcionales que integran la biblioteca de componentes de interfaz de usuario.

<b>Número:</b> 1	<b>Nombre del requisito:</b> Validar formato de una cadena de caracteres en JFormateadorFX.	
<b>Programador:</b> Greter Gómez Martínez Anchel Durán Bolaño	<b>Iteración Asignada:</b> 1	
<b>Prioridad:</b> Alta	<b>Tiempo Estimado:</b> 3 días	
<b>Riesgo en Desarrollo:</b> Medio	<b>Tiempo Real:</b> 2 días	
<b>Descripción:</b> Permite validar la entrada de caracteres al campo del texto mediante la inserción o selección de expresiones regulares en las propiedades del componente. Las expresiones regulares se encuentran definidas para la validación de nombres, apellidos, iniciales, números		



## Requisitos no Funcionales

Los requisitos no funcionales imponen restricciones al diseño o funcionamiento del sistema. Generalmente son aplicados al sistema en su totalidad y no se refieren directamente a funciones específicas, sino a características del sistema; la usabilidad y la eficiencia son algunos de ellos. Pueden estar dirigidos incluso al proceso de desarrollo, especificando herramientas o estándares de calidad a emplear en función de las necesidades del usuario (Sommerville, 2007). A continuación se muestra el listado de los requisitos no funcionales:

### Usabilidad

RnF-1. La biblioteca de componentes de interfaz de usuario podrá ser utilizada por cualquier desarrollador con conocimientos básicos de Java y JavaFX.

RnF-2. Los nombres de las propiedades de los componentes de interfaz de usuario se establecerán de manera que resulte de fácil comprensión para los desarrolladores.

### Mantenibilidad

RnF-3. La aplicación debe permitir que se le incorporen nuevas funcionalidades.

### Portabilidad

RnF-4. La aplicación constituirá una biblioteca de componentes de interfaz para un entorno de escritorio.

RnF-5. La aplicación debe ser multiplataforma.

### Interfaz

RnF-6. Interfaz accesible e intuitiva, el manejo de los componentes de interfaz de usuario debe ser lo más intuitivo posible, de manera que sean muy claras las acciones que se puedan realizar con ellos.

### Requisitos de software

RnF-7. Proporcionar características mínimas de software a las estaciones de trabajo.

Las características mínimas de software deben ser las siguientes:

- ✓ Utilizar para el soporte de las librerías de Java y JavaFX a partir de la JDK 8 en adelante.
- ✓ Utilizar para el desarrollo de las interfaces gráficas la herramienta JavaFX *Scene Builder* a partir de la versión 2.0 en adelante para la integración de los componentes.
- ✓ Utilizar para el desarrollo de los componentes la herramienta IDE Netbeans a partir de la versión 8.0.

## Requisitos de hardware

RnF-8. Proporcionar características mínimas de hardware a las estaciones de trabajo.

Las características técnicas mínimas de hardware deben ser las siguientes:

- ✓ 1 GB de RAM
- ✓ 30 GB de disco duro
- ✓ 2.0 GHz de procesamiento

Al concluir el levantamiento de los requisitos, tanto funcionales como no funcionales, se aplicaron técnicas para validar la obtención de los requisitos. Estos resultados alcanzados en la validación de los requisitos pueden ser consultados en el Capítulo 3.

## 2.3 Diseño

Durante esta disciplina se modela el diagrama de clase de los componentes de la biblioteca para que soporten los requisitos. Contribuyendo a una comprensión precisa, de fácil descripción y ayude a la estructuración del software incluyendo su arquitectura. Además, en esta disciplina se desarrolló: el modelo de diseño el cual lo componen los diagramas de clases y el diagrama de paquete.

### Diseño Arquitectónico

La arquitectura de software que presenta la biblioteca de componentes de interfaz gráfica de usuario, se desarrolló sobre la plataforma JavaSE, basándose en la arquitectura MVC; la cual se empleó con el objetivo de proporcionar una correcta interacción entre el modelo y la vista del componente. El uso de las especificaciones de JavaFX Beans, permitió la integración de los componentes de la biblioteca de interfaz gráfica en la herramienta de diseño JavaFX Scene Builder. Esta integración conllevó a implementar las funcionalidades de los componentes, para cumplir las necesidades de los desarrolladores utilizando las clases, objetos, eventos y propiedades de la tecnología JavaFX de forma organizada. Además, permitió reutilizar clases de JavaFX desarrolladas previamente, entre las que se pueden mencionar algunas como TableView, TextField, VBox, HBox entre otras.

En el diagrama de clases (Imagen 3), se pone en evidencia la interacción de los elementos de la arquitectura MVC. En la clase JFormateadorFX, se puede observar el elemento Modelo mediante los datos o estados del componente, como ejemplo de ellos se encuentran listas de objetos y datos que contenga el componente, representando la información con la cual el componente opera. Por tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. La Vista se evidencia mediante la clase JFormateadorFXML, que es la encargada de representar el visual de la GUI. Permite determinar qué eventos pasan al Controlador para interactuar con la interfaz de usuario.



El controlador se pone de manifiesto mediante la relación de clases entre el `JFormateadorFXController`, `JFormateadorFX` y `EExpresiones`, este elemento se encarga de recoger las acciones del usuario, interactúa con el modelo y devuelve una determinada vista en respuesta. El Controlador responde a eventos (usualmente acciones del usuario final) e invoca peticiones al Modelo cuando se hace alguna solicitud sobre la información, como por ejemplo, registrar o editar una información. Siendo de esta forma el intermediario entre la Vista y el Modelo. Todos los componentes de la biblioteca, utilizan esta estructura de sistema de clases, quedando definido así por el sistema de clases que emplea JavaFX para su funcionamiento.

### Diagrama de clases

Un diagrama de clases es una descripción de las clases en un sistema y sus relaciones. No describe el comportamiento dinámico del sistema, sino la estructura del mismo mostrando sus clases, atributos y relaciones entre ellos (Pressman, 2005).

El `JFormateadorFX` es un componente que mediante expresiones regulares realiza validaciones al texto introducido por el usuario. Este componente está compuesto por cinco clases las cuales se describen a continuación.

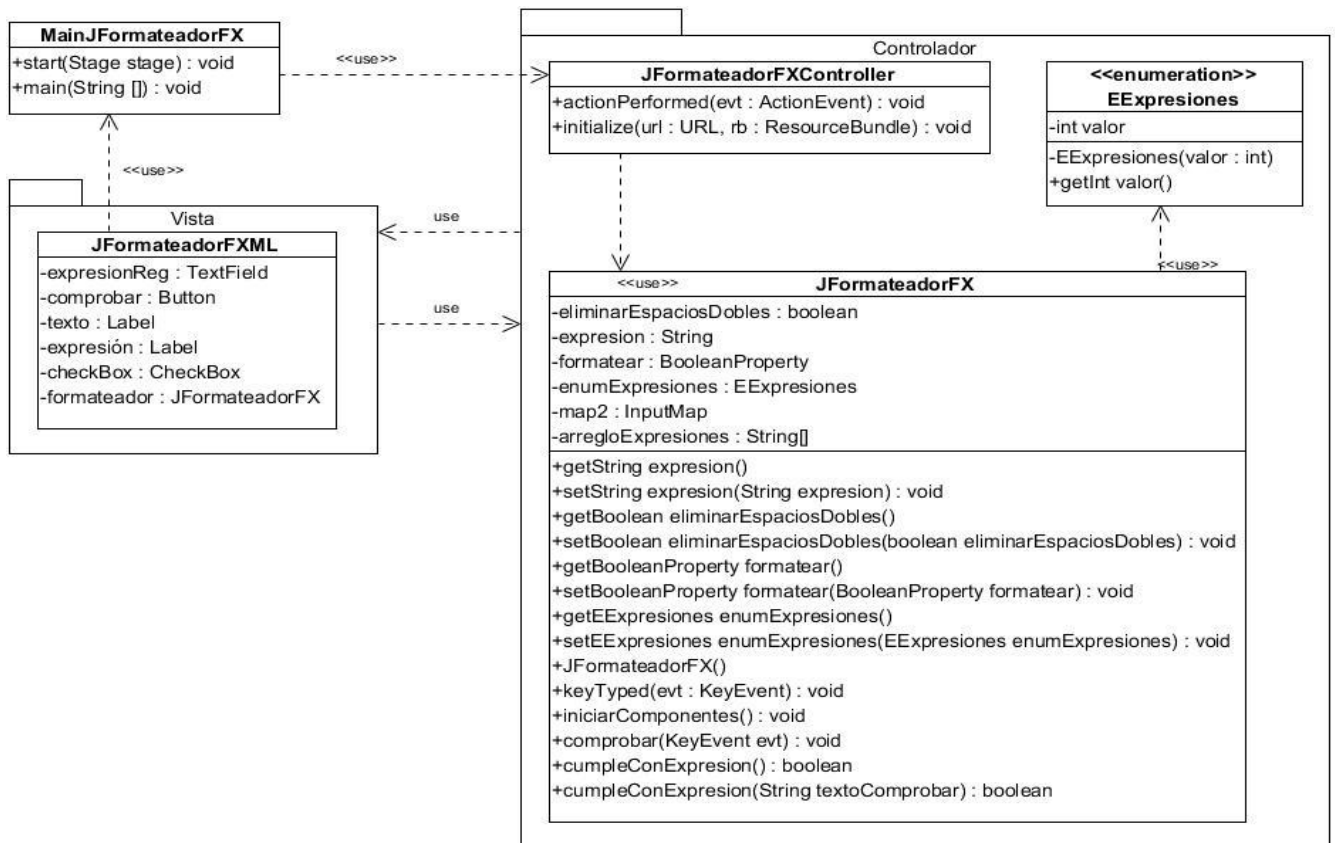


Imagen 3. Diagrama de clases del componente `JFormateadorFX`.

Seguido, se muestran las tablas de descripción de cada una de las clases que conforman al componente JFormateadorFX. Estas tablas son utilizadas para describir el objetivo principal de cada clase y para describir los métodos contenidos en estas.

Nombre de la clase: <b>JFormateadorFX</b>	
<b>Descripción:</b> Clase que contiene las funcionalidades del componente y se apoya en la clase EExpresiones para realizar su funcionamiento.	
Método	Descripción
getString expresión()	Devuelve el valor de la expresión regular definida.
setString expresion(String expresion)	Modifica el valor de la expresión regular.
getBoolean eliminarEspaciosDobles()	Si el valor del atributo es true elimina los espacios dobles de una cadena de caracteres.
setBoolean eliminarEspaciosDobles(boolean eliminarEspaciosDobles)	Modifica el valor del atributo eliminarEspaciosDobles().
getBooleanProperty formatear()	Si el valor del atributo es true admite las restricciones del formateador de acuerdo a la expresión regular.
setBooleanProperty formatear(BooleanProperty formatear)	Modifica el valor del atributo formatear().
getEExpresiones enumExpresiones()	Devuelve el enumerador de la expresión regular seleccionada.
setEExpresiones enumExpresiones(EExpresiones enumExpresiones)	Permite modificar el valor de la expresión regular seleccionada.
JFormateadorFX()	Constructor.
keyTyped(evt:javafx.scene.input.KeyEvent)	Obtiene el evento de entrada del teclado.
iniciarComponentes()	Inicializa el evento de teclado del método comprobar.
comprobar()	Valida que no entren caracteres al formateador que no cumplan con la expresión regular definida.
cumpleConExpresion()	Comprueba si la expresión regular introducida es correcta o no.

**Tabla 5. Descripción de la clase JFormateadorFX.**

Nombre de la clase: <b>JFormateadorFX</b>	
<b>Descripción:</b> Clase de tipo Enumerador que contiene expresiones regulares definidas.	
Método	Descripción
EExpresiones(valor Int)	Constructor.
getInt valor()	Devuelve el valor entero del enumerador.

**Tabla 6. Descripción de la clase EExpresiones.**

Nombre de la clase: <b>JFormateadorFXController</b>	
<b>Descripción:</b> Es la clase controladora del formateador.	
Método	Descripción
actionPerformed(evt ActionEvent)	Ejecuta la acción de un botón.
initialize()	Inicializa los atributos y métodos del componente.

**Tabla 7. Descripción de la clase JFormateadorFXController.**

Nombre de la clase: <b>MainJFormateadorFX</b>	
<b>Descripción:</b> Es la clase que especifica dónde comienza la ejecución del programa.	
Método	Descripción
start(Stage stage)	Inicia una escena.
main(String [])	Inicializa los atributos y métodos de la clase.

**Tabla 8. Descripción de la clase MainJFormateadorFX.**

Nombre de la clase: <b>JFormateadorFXML</b>	
<b>Descripción:</b> Es la clase que inicializa los componentes visuales en FXML.	

**Tabla 9. Descripción de la clase JFormateadorFXML.**

### **Patrones de diseño empleados en la solución**

El correcto uso de los patrones de diseño para la generación de los artefactos necesarios, facilitó crear un punto de partida a las actividades de implementación, con el fin de brindar una solución a los problemas de desarrollo de la biblioteca de componentes. Los patrones descritos a continuación se utilizan en todos los componentes de la biblioteca, debido a que cada componente es individual y emplean la misma forma del sistema de clases.

## Patrones GRASP

**Controlador:** define qué clase debe responder a determinados eventos. Como ejemplo de este patrón se puede observar en el componente JFormateadorFX y se evidencia en la clase JFormateadorFXController, la cual es la encargada de todos los eventos y funcionalidades con que cumple el componente.

**Experto:** consiste en asignar una responsabilidad a la clase experta en información. Como ejemplo de este patrón se puede observar en el componente JFormateadorFX y se evidencia en la clase JFormateadorFX, la cual cuenta con la información necesaria para cumplir las responsabilidades del componente.

**Creador:** este patrón designa la clase que guía la asignación de responsabilidades relacionadas con la creación de objetos. Como ejemplo de este patrón se puede ver en el componente JFormateadorFX y se evidencia en la clase MainJFormateadorFX que es la encargada de crear y definir los objetos necesarios para la inicialización del componente.

**Bajo acoplamiento:** se encarga de mantener las clases lo menos ligadas entre sí posible. Como ejemplo de este patrón se puede observar en el componente JFormateadorFX y se evidencia en la clase JFormateadorFXML la cual si se produce una modificación, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

**Alta cohesión:** se basa en que la información que almacena una clase sea coherente y esté en la medida de lo posible, relacionada con la clase. Como ejemplo de este patrón se observa en el componente JFormateadorFX y se evidencia en la clase JFormateadorFXController logrando un incremento de la claridad y comprensión, lo que simplifica el mantenimiento.

## Patrones GoF

**Estructurales:** se utiliza el patrón Fachada para proveer una interfaz al código más fácil de utilizar.

**Creacionales:** se utiliza el patrón Factoría para convertir tipo de datos a datos genéricos y viceversa, con el objetivo de ocultar detalles de cómo son creados y manipulados los objetos.

**Comportamiento:** se utiliza el patrón Estado para permitir que eventos de datos de un objeto, modifique su comportamiento cada vez que haya un evento de cambio de estado.

La actividad de Diseño fue esencial en el proceso de desarrollo de software. Se realizó con el objetivo de permitir la implementación de la biblioteca de componentes de interfaz gráfica de usuario de manera

precisa. La descripción de las actividades realizadas durante la implementación se presenta en el epígrafe que aparece a continuación.

## 2.4 Implementación

Esta disciplina está enfocada a la implementación de la biblioteca de componentes de interfaz gráfica de usuario a partir de los resultados del diseño. Se muestra el diagrama de componentes para apoyar la implementación y se especifican los estándares de codificación para la posterior estructuración del código. Además, se describen las funcionalidades de cada uno de los componentes que conforman la biblioteca de interfaz gráfica de usuario.

### Diagrama de Componentes

En el presente diagrama se modela la estructura de la biblioteca de interfaz gráfica, además, se muestran los componentes que la conforman y las dependencias de las clases que brinda JavaFX y los componentes de la biblioteca.

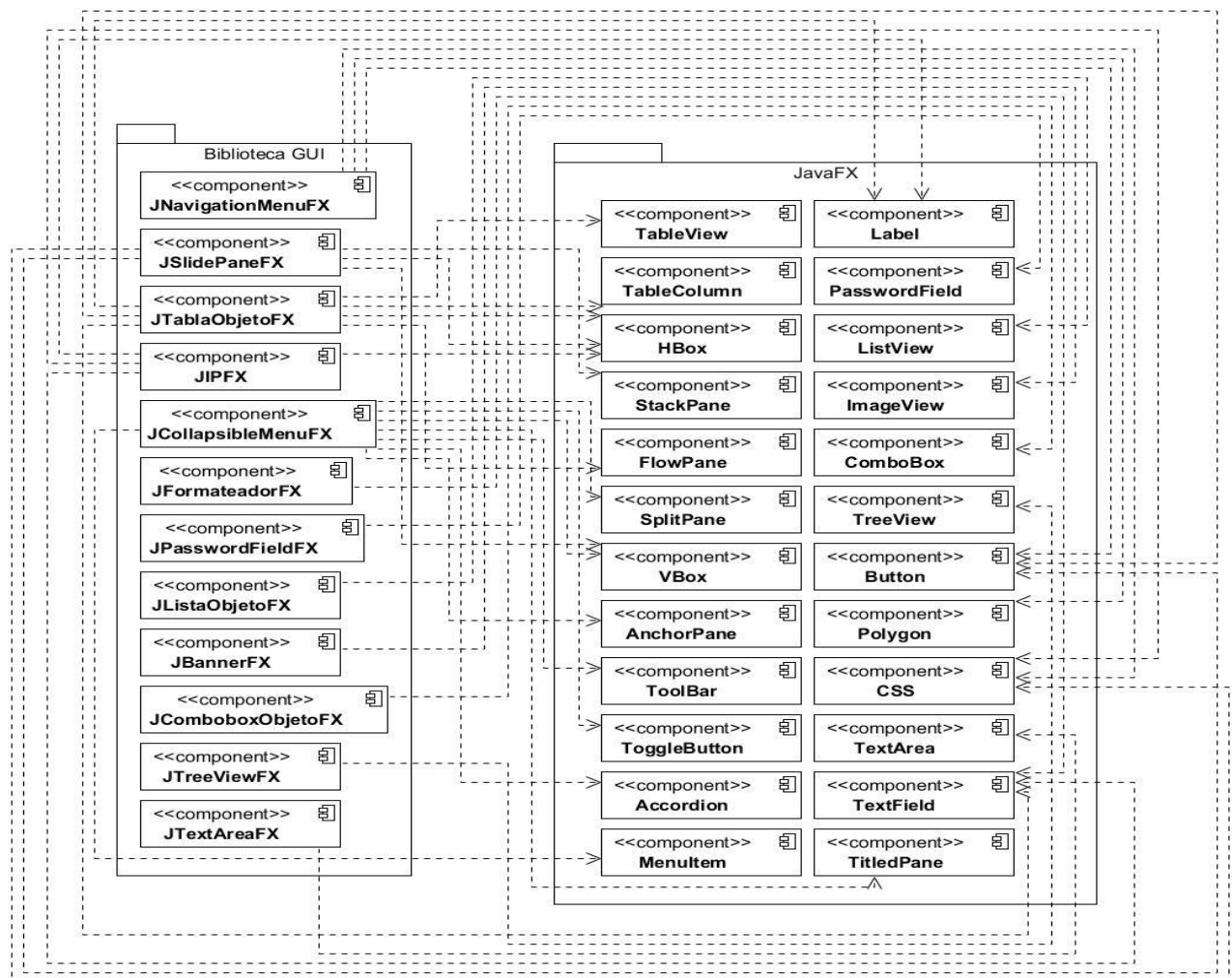


Imagen 4. Diagrama de Componentes.

## Estándares de codificación

Para el desarrollo de la biblioteca de componentes de interfaz gráfica de usuario se utilizó el estándar de codificación para Java, definido por la Dirección Técnica de la Producción basado en Convenciones de Código para el lenguaje de programación Java por Scott Hommel Sun *Microsystems Inc.* Determinado por la UCI en el año 2010 para proyectos desarrollados en el lenguaje de programación Java. Estos estándares de codificación pueden ser consultados en (Producción, 2011).

Estos estándares de codificación establecieron las características a cumplir para la implementación de la biblioteca de componentes. Dentro de las normativas presentadas se encuentran las extensiones de los ficheros que pueden ser para Fuente Java (.java) y Bytecode de Java (.class). Otras de estas reglas son la organización de los ficheros que está compuesto por secciones que deben estar separadas por líneas en blanco y comentarios opcionales que identifican cada sección, deben ser evitados los ficheros de más de 2000 líneas. Las demás especificidades pueden ser consultadas en (Producción, 2011).

Luego de establecidos los estándares de codificación se dio paso a la descripción de cada uno de los componentes de la biblioteca de interfaz gráfica de usuario.

## Componentes de Interfaz de Usuario

### JFormateadorFX

Es un componente de entrada de texto que permite al usuario introducir una sola línea de texto sin formato. 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 JFormateadorFX. 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:

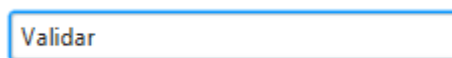


Imagen 5.Componente JFormateadorFX.

### JComboBoxObjetoFX

Un ComboBox es un componente que muestra una lista desplegable a los usuarios que les proporcionan una opción que pueden elegir, también permite la entrada textual de información. JavaFX cuenta con un ComboBox dentro de sus componentes, este contiene funcionalidades que facilitan su

uso en las aplicaciones. JComboBoxObjetoFX es un ComboBox 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 se quieran mostrar, a este componente sólo es necesario insertarle la lista de objetos y señalarle qué atributo debe ser mostrado. A continuación se muestra una imagen del componente:

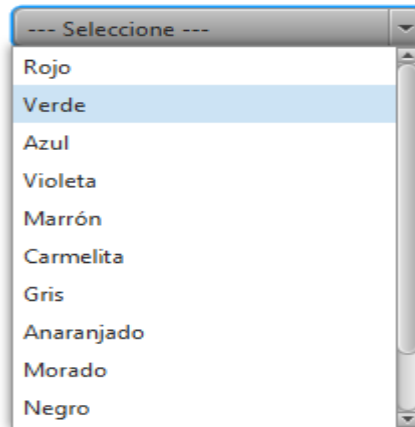


Imagen 6.Componente JComboBoxObjetoFX.

### JCollapsibleMenuFX

Un JCollapsibleMenuFX es un componente que muestra un menú plegable con la estructura de los flujos de trabajo por los que debe transitar un usuario, este menú toma las funcionalidades de los flujos de trabajo en dependencia de los permisos que tengan otorgado los usuarios registrados en la aplicación en tiempo de ejecución.

Su principal función es que al plegar o desplegar el menú, permita que el área de trabajo cuente con un mayor espacio para mostrar los componentes que presentan la información a los usuarios finales. También permiten que los elementos que conforman el menú presenten orden por jerarquías atendiendo al orden de las informaciones.

JCollapsibleMenuFX está compuesto por varios componentes de JavaFX:

- ✓ **ToolBar**: para almacenar el botón o los botones que ejecuta la acción de plegar o desplegar el menú, entre otras funciones.
- ✓ **ToggleButton**: para ejecutar la acción de plegar o desplegar el menú, mostrando en dos tiempos la acción, en el primer tiempo el botón se encuentra seleccionado indicando que el menú se encuentra plegado y en el segundo tiempo se encuentra deseleccionado indicando que el menú se encuentra desplegado.

- ✓ TitlePane: para mostrar el nombre de los elementos contenidos dentro del menú que se encuentran seleccionados.
- ✓ VBox: para almacenar los MenuButton o Button que compongan el flujo de trabajo del menú.
- ✓ MenuButton o Button: para conformar el flujo de trabajo del menú.
- ✓ SplitPane: para contener los componentes del menú.

A continuación se muestra la imagen del componente:

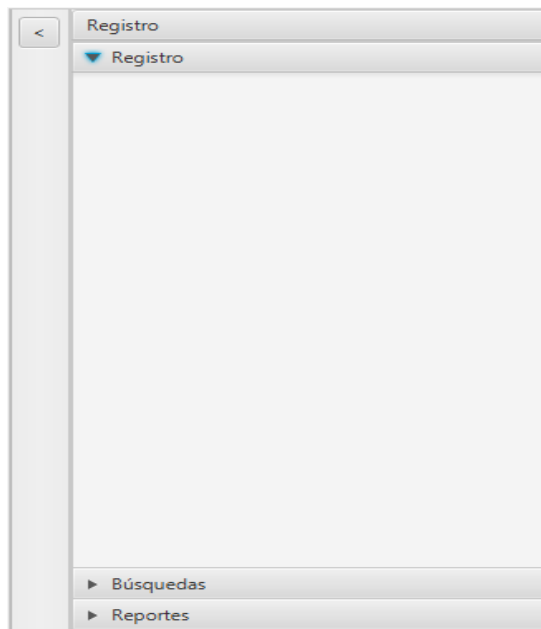


Imagen 7.Componente JCollapsibleMenuFX.

### JTablaObjetoFX

Se integra JTablaObjetoFX a la biblioteca de componentes realizada en JavaFX. Este componente es uno de los más importantes debido a su uso para mostrar diversas informaciones de forma organizada, 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 JTablaObjetoFX se le debe insertar datos a una ObservableList de instancias de clases propias de las aplicaciones que se quieran mostrar. Se definen los atributos que se deseen mostrar en la tabla y actualizándole la lista del componente, automáticamente cualquier cambio que se realice en la tabla se ajustará en la vista de la tabla mostrada al usuario.

Este componente agrega dos funcionalidades especiales que facilitan el trabajo con la tabla. En la parte superior, muestra una búsqueda que permite filtrar todos los datos que contengan la expresión escrita en el texto de filtrado. También incluye un paginado en la parte inferior, que permite mostrar una vista de la tabla con la cantidad de elementos que especifique el usuario y la página en que se encuentra.



JTablaObjetoFX permite una configuración especial para que en cada una de sus columnas muestre los atributos de las instancias de la lista (los que el desarrollador le indique). Además, presenta muchas más funcionalidades que no contiene un JTable común de la librería Swing. A continuación se muestra una imagen del componente:

Filtrar Búsqueda: <input type="text"/>			
Nombre	Edad	Sexo	Nota
Anchel	25	Masculino	5.0
Greter	22	Femenino	5.0
Fernando	30	Masculino	4.5
Mirtha	21	Femenino	4.0
Robin	26	Masculino	3.0
María	20	Femenino	3.5
Martha	27	Femenino	3.2
Carlos	28	Masculino	4.2
Ana	21	Femenino	5.0
Roberto	26	Masculino	4.0

**Imagen 8.Componente JTablaObjetoFX.**

## JIPFX

JavaFX no cuenta con un componente para la gestión de direcciones IP, es por eso que surge JIPFX para manejar estas direcciones con exactitud. JIPFX trae implementado un gran número de restricciones que aseguran un correcto manejo de las direcciones IP. Este componente está conformado por cuatro cuadros de texto que son los que permiten la entrada de los números correspondientes a la configuración de la red. Cada cuadro de texto permite la entrada solamente de uno a tres números y a la vez valida que cada número esté en un rango entre 0 y 255 que son los rangos admisibles con que debe 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 se asignan el valor cero para que no ocurran errores en la numeración IP. A continuación se muestra una imagen del componente:

10 . 52 . 5 . 89

**Imagen 9.Componente JIPFX.**

## JNavigationMenuFX

Es un componente que permite mostrar al usuario el camino que está siguiendo dentro de la aplicación y dónde se encuentra navegando el usuario en ese momento. Está basado en la técnica migas de pan que se utiliza para mejorar la navegabilidad en la interfaz del usuario. Cada elemento que conforma el camino es un enlace a una nueva sección, de esta forma el usuario encuentra más cómoda la navegación dentro de la aplicación. JNavigationMenuFX es un componente compuesto por un panel que organiza los componentes hijos de forma horizontal y por los botones que conforman el flujo del camino. Los botones ofrecen al usuario navegar hasta el panel deseado mediante un evento de clic, brindándole como ventaja mejora en la usabilidad de la aplicación. El usuario puede ir de un lado al otro, dando saltos intermedios entre una categoría principal y las subcategorías, lo que agiliza de manera considerable el tiempo que tarda en alcanzar el panel que desea ver. A continuación se muestra una imagen del componente:



Imagen 10.Componente JNavigationMenuFX

## JListaObjetoFX

Es un componente que muestra una lista vertical de los elementos que el usuario puede seleccionar, o con los que pueda interactuar. Es capaz de tener su tipo genérico de representar los tipos de datos en el modelo de respaldo. Hacer esto tiene la ventaja de hacer diversos métodos en el JListaObjetoFX así como las clases de apoyo. Además, haciendo uso de los soportes genéricos se simplifica considerablemente el desarrollo de aplicaciones que hacen uso del componente. JListaObjetoFX admite instancias de cualquier clase y en su listado de información, muestra los atributos de instancias que le sean indicados en las propiedades particulares del componente. 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:



Imagen 11.Componente JListaObjetoFX.

## JBannerFX

Un banner, también conocido como cartel de anuncio, es un anuncio en forma de texto o imagen normalmente rectangular colocada arriba, debajo o en los lados del contenido principal de un sitio web o aplicación de escritorio. JavaFX propone el componente JBannerFX, el cual brinda la posibilidad de mostrar un banner para las aplicaciones de escritorio. La ventaja que brinda este componente es que permite que al redimensionar la aplicación no se distorsione la imagen del banner. Esta ventaja está dada por el componente ImageView que compone JavaFX. Este componente permite cambiar el tamaño de la imagen que se muestra (con o sin conservar la relación de aspecto original de la imagen) y la especificación de una ventana en la imagen de origen para restringir los píxeles mostrados por este componente. A continuación se muestra una imagen del componente:



Imagen 12.Componente JBannerFX.

## JPasswordFieldFX

Es uno de los componentes con que cuenta la librería de JavaFX. Este componente es usado para la entrada de contraseñas de los usuarios. JPasswordFieldFX es una subclase de TextField, pero en vez de mostrar el carácter real tecleado por el usuario, muestra otro carácter, el cual puede ser o no modificado por el usuario; esto tiene como objetivo ocultar la identidad de la contraseña del usuario para que no pueda ser vista por otras personas. JPasswordFieldFX es también usado para la entrada de contraseñas, pero con características visuales más agradables. 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:

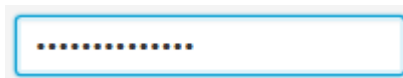


Imagen 13.Componente JPasswordFieldFX.

## JTextAreaFX

Es un componente de entrada de texto que permite al usuario introducir varias líneas de texto sin formato. Esta característica especial, hace que este componente sea una buena elección a la hora de requerir comentarios o mensajes. Si una cadena de texto supera el ancho de la dimensión del JTextAreaFX, entonces el componente indicará si el texto debe pasar a otra línea. También permite la

validación de cualquier texto que sea entrado por el usuario haciendo uso de las expresiones regulares que son insertadas en una de sus propiedades. JTextAreaFX puede personalizar el área de texto de varias maneras, aunque un área de texto se puede mostrar en una sola fuente y color, permite establecer qué tipo de letra y color utiliza. Mediante los eventos de teclado Ctrl + N el componente es capaz de cambiar el formato de los caracteres de tipo normal a negrita y viceversa si se vuelve a presionar el mismo evento, también presenta el evento Ctrl + K encargado de convertir los caracteres de tipo normal a cursiva y viceversa. A continuación se muestra una imagen del componente:

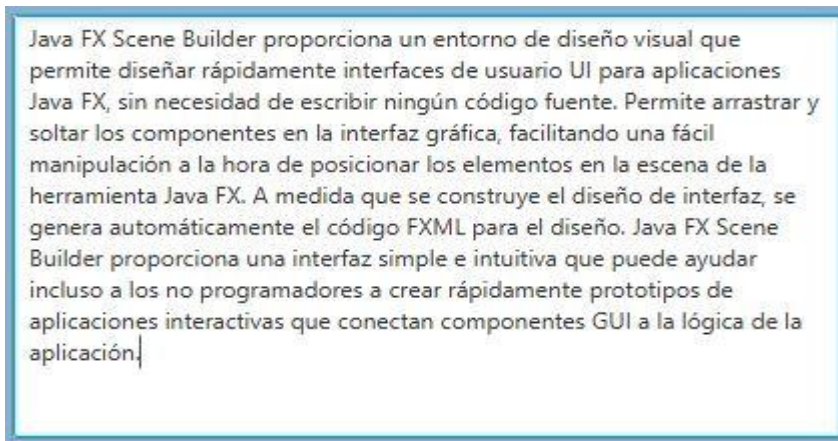


Imagen 14.Componente JTextAreaFX.

### JSlidePaneFX

Es un componente compuesto por uno o más paneles deslizables que permite mostrar los paneles que contienen la información del marco de trabajo. Este componente permite mediante los botones de navegación o wizard<sup>20</sup> mostrar al usuario una vista de la información que contienen los paneles dentro del marco de trabajo. JSlidePaneFX está compuesto por un panel vertical, el cual contiene en la parte superior de su estructura un Panel Pila (del inglés StackPane) que almacena y muestra sus hijos, es decir muestra los paneles que contienen la información en el orden de una pila. Además, presenta un panel horizontal en la parte inferior compuesto por botones que manipulan el flujo de los paneles a mostrar. Estos botones permiten desplazar los paneles hacia adelante y hacia atrás según la acción del usuario, también presentan la funcionalidad especial que les indica cuándo habilitarse o deshabilitarse en tiempo de ejecución, haciendo más intuitivo la navegación del usuario dentro de la aplicación. Estos botones pueden presentar las funcionalidades siguientes:

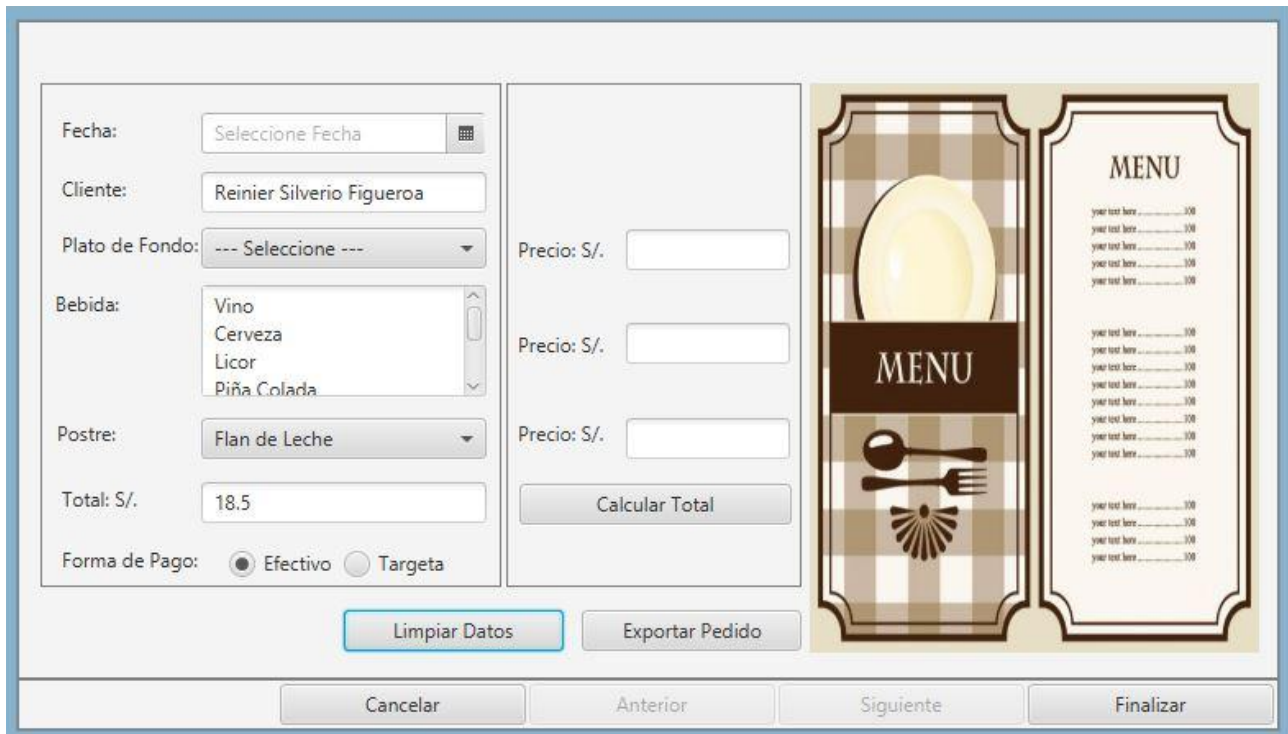
- ✓ Botón Cancelar: permite terminar un proceso en ejecución.
- ✓ Botón Anterior: permite desplazar al panel anterior.

---

<sup>20</sup> Secuencia de cuadros de diálogo que llevan al usuario a través de una serie de pasos bien definidos.

- ✓ Botón Siguiente: permite desplazar al panel siguiente.
- ✓ Botón Finalizar: permite concluir un proceso en ejecución guardando todos los cambios realizados en el paso del flujo de trabajo.

Además, el usuario puede añadir o modificar el panel que contiene los botones a voluntad propia en caso de ser necesario durante el desarrollo. A continuación se muestra una imagen del componente:



The screenshot displays a Java Swing application window with a light blue border. On the left side, there is a form for entering order details. The fields include:
 

- Fecha:** A date selection field with a calendar icon.
- Cliente:** A text field containing "Reinier Silverio Figueroa".
- Plato de Fondo:** A dropdown menu showing "--- Seleccione ---".
- Bebida:** A list box containing "Vino", "Cerveza", "Licor", and "Piña Colada".
- Postre:** A dropdown menu showing "Flan de Leche".
- Total: S/.** A text field containing "18.5".
- Forma de Pago:** Radio buttons for "Efectivo" (selected) and "Targeta".

 In the center, there are three price input fields labeled "Precio: S/." and a "Calcular Total" button. Below these are "Limpiar Datos" and "Exportar Pedido" buttons. At the bottom of the form area are "Cancelar", "Anterior", "Siguiente", and "Finalizar" buttons. On the right side, there is a preview of a menu card with a yellow plate, silverware, and a menu list titled "MENU" with several items and prices.

**Imagen 15.Componente JSlidePaneFX.**

### **JTreeViewFX**

Es un componente que proporciona una vista de una lista de elementos en forma de árbol. Permite profundizar en los hijos de un elemento del árbol (treeitem) de forma recursiva hasta que un treeitem no tiene hijos (es decir, que es un nodo hoja en el árbol). Para facilitar esto, a diferencia de los componentes como JComboboxObjetoFX, JTablaObjetoFX y JListaObjetoFX, en JTreeViewFX es necesario sólo especificar el nodo raíz y el nodo por el cual se agruparán los elementos de la lista. Este componente muestra el árbol en tres niveles, nodo raíz, nodos de agrupación y nodos hojas mediante las propiedades especiales que le permite especificar qué elementos de la lista mostrar. También JTreeViewFX permite personalizar los nodos contenidos en él con imágenes que muestren qué elementos son directorios o archivos, presentando una vista más agradable de sus nodos hijos. A continuación se muestra una imagen del componente:



Imagen 16.Componente JTreeViewFX.

### Conclusiones Parciales

- ✓ El modelado de negocio permitió comprender el proceso de informatización de la biblioteca de componentes de interfaz de usuario.
- ✓ La descripción de la biblioteca de componentes permitió conocer las ventajas del uso de la tecnología JavaFX en la realización de los componentes de interfaz gráfica.
- ✓ La utilización de estrategias de captura de requisitos como: Tormenta de Ideas y Entrevistas permitió realizar la definición, especificación y descripción de los requisitos de la biblioteca de componentes.
- ✓ El diseño de la biblioteca de componentes permitió realizar los diagramas de clases, la estructura y relaciones entre las clases que se manejan en los componentes.
- ✓ La utilización de patrones propició la confección de diagramas de clases de los componentes.
- ✓ El diagrama de componentes y el diagrama de paquetes facilitaron la disciplina de implementación.
- ✓ La utilización de los estándares de implementación, facilitó el entendimiento del código por los desarrolladores y el futuro mantenimiento de los componentes de la biblioteca de interfaz gráfica.

## VALIDACIÓN DE LA INVESTIGACIÓN

# 3

*Cualquier tecnología suficientemente avanzada es indistinguible de la magia.*

*Arthur C. Clarke*

### Introducción

El presente capítulo aborda las validaciones correspondientes a las disciplinas propuestas por la metodología de desarrollo como buenas prácticas del desarrollo de software, así como la validación de las variables de la investigación.

### 3.1 Técnicas y métricas para validar los requisitos y el diseño

Las técnicas y métricas que se utilizaron para la validación de los requisitos identificados fueron las siguientes:

#### Resultados de la técnica Construcción de prototipos

Cada uno de los prototipos le permitió a los especialistas tener una idea de las interfaces gráficas de los componentes. Estos se realizaron de forma funcional con la herramienta Netbeans con el objetivo de lograr mayor aprobación por parte del cliente.

#### Resultados de la técnica Revisión

Con esta técnica se validó que la interpretación de cada una de las descripciones no sea ambigua, ni presenten omisiones o errores y además, cada uno de los requisitos cumpla con lo que necesita el cliente.

#### Resultados de la métrica especificidad de requisitos

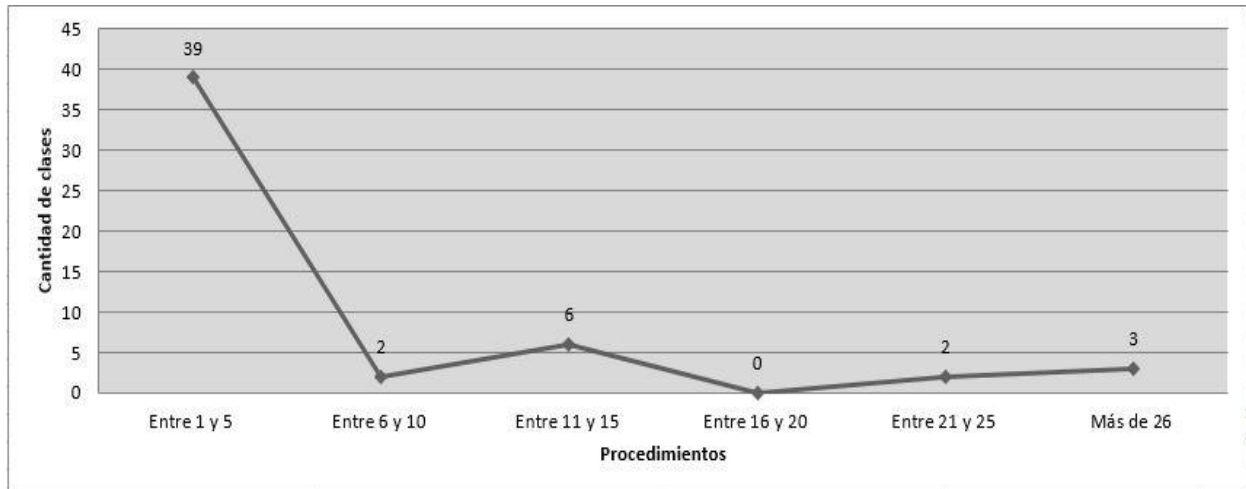
Para medir la ausencia de ambigüedad de los requisitos, se aplicó la métrica especificidad de requisitos:

$$Q_1 = \frac{67}{70} = 0.957$$

Como resultado de la aplicación de la métrica, se obtuvo un valor de 0.957, cifra que se aproxima al valor uno, por tanto, mayor será la consistencia de la interpretación de los revisores para cada requisito y menor será la ambigüedad en la especificación de los requisitos.

#### Resultados de la métrica Tamaño Operacional de Clases

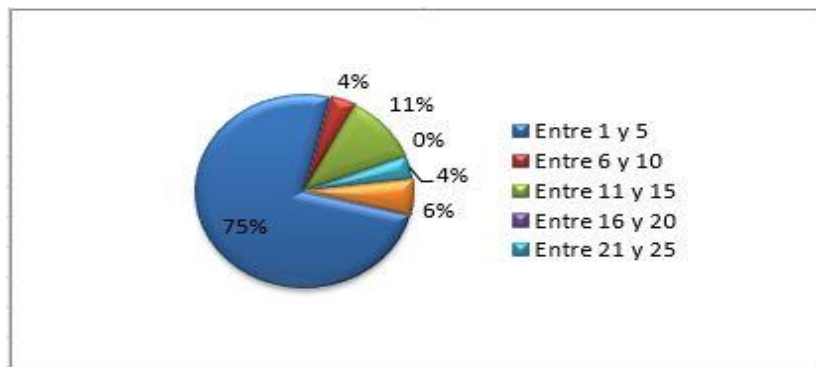
Está dado por el número de métodos asignados a una clase. En la aplicación de esta métrica se tuvieron en cuenta las clases que contienen cada uno de los componentes. Se tuvo en cuenta además, la cantidad de procedimientos que contienen cada una de las clases, obteniendo los siguientes resultados:



**Imagen 17. Representación de la cantidad de clases y número de procedimientos.**

Para ver el modelo de medición utilizado en la métrica TOC (ver Anexo 4).

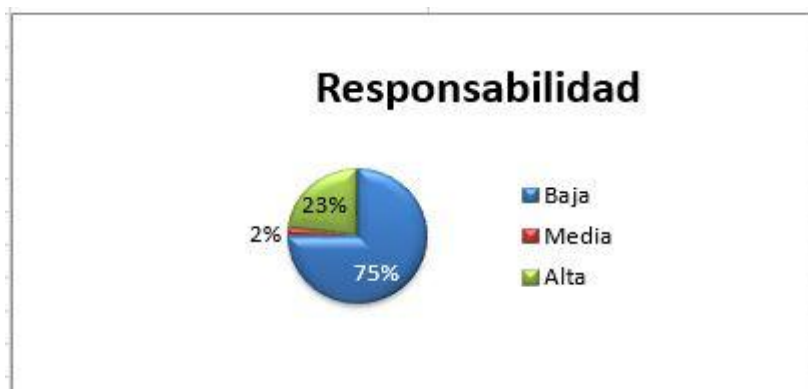
Representación en por ciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos:



**Imagen 18. Representación en por ciento de la cantidad de clases y número de procedimientos.**

Representación en por ciento de la incidencia de los resultados obtenidos en el atributo.

**Responsabilidad:**



**Imagen 19. Representación en por ciento del valor del atributo Responsabilidad.**



Representación en por ciento de la incidencia de los resultados obtenidos en el atributo

**Complejidad de implementación:**



**Imagen 20. Representación en por ciento del valor del atributo Complejidad de implementación.**

Representación en por ciento de la incidencia de los resultados obtenidos en el atributo.

**Reutilización:**



**Imagen 21. Representación en por ciento del valor del atributo Reutilización.**

Análisis de los resultados obtenidos en la evaluación de la métrica TOC:

Se puede concluir que el diseño de la Biblioteca de Componentes de Interfaz gráfica alcanzó resultados positivos durante la evaluación de la métrica. Se obtuvo un 75% de baja responsabilidad de las clases y un 75% de complejidad de implementación, mientras que la reutilización es alta para un 80%. Estos datos favorecen el buen uso del diseño empleado y el resultado de la implementación.

### 3.2 Verificación de la implementación

La verificación de la implementación se realizó mediante las disciplinas pruebas internas y de liberación que propone la metodología Variación AUP-UCI.

#### Resultados del método de prueba de caja blanca

Fue necesario calcular antes la complejidad ciclomática<sup>21</sup> del algoritmo o fragmento de código a analizar. A continuación se muestra el grafo de flujo asociado al método y se enumeran las sentencias de código del procedimiento realizado sobre uno de los eventos del método `Paginado()` del componente `JTablaObjetoFX` en la clase `JTablaObjetoFX` (ver Imagen 30).

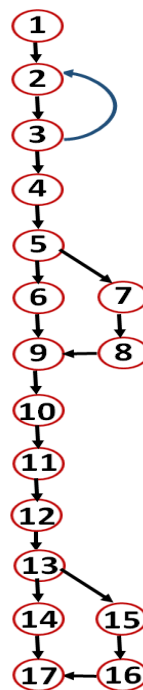


Imagen 22. Grafo de flujo asociado al método `Paginado()`.

<sup>21</sup> Número de caminos independientes en un programa.

```

private void Paginado() {
    izquierda.addEventHandler(ActionEvent.ACTION, new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent event) {
            numFilas = Integer.parseInt(numero.getText()); //1
            derecha.setDisable(false); //1
            List l = new ArrayList<ObservableList>(); //1
            for (int i = 0; i < getTablaLista().size(); i++) { //2
                l.add(getTablaLista().get(i)); //3
            }
            var1 -= numFilas; //4
            if (var1 > getTablaLista().size()) { //5
                tabla.getItems().clear(); //6
                tabla.getItems().addAll(l.subList(var1 - numFilas, getTablaLista().size())); //6
            } else if (var1 >= 0) { //7
                tabla.getItems().clear(); //8
                tabla.getItems().addAll(l.subList(var1 - numFilas, var1)); //8
            } else { //9
                izquierda.setDisable(true); //10
            }
            if (var1 - numFilas <= 0) { //11
                izquierda.setDisable(true); //12
            }
            if ((getTablaLista().size() % numFilas) != 0) { //13
                pag.setText((numPag - 1) + " de " + (getTablaLista().size() / numFilas + 1) + " Páginas"); //14
            } else { //15
                pag.setText((numPag - 1) + " de " + (getTablaLista().size() / numFilas) + " Páginas"); //16
            }
            numPag -= 1; //17
        }
    });
}

```

Imagen 23. Complejidad ciclomática del método Paginado().

Cálculo de la complejidad ciclomática

$V(G) = (A - N) + 2$  Donde "A" es la cantidad de aristas y "N" la cantidad de nodos.

$$V(G) = (19-17) + 2 = 4$$

$V(G) = P + 1$  Siendo "P" la cantidad de nodos predicados (nodos donde parten  $A \geq 2$ ).

$$V(G) = 3 + 1 = 4$$

$V(G) = R$  Donde "R" representa la cantidad de regiones en el grafo.

$$V(G) = 4$$

El cálculo efectuado mediante las fórmulas ha dado el mismo valor, por lo que se puede plasmar que la complejidad ciclomática del código es de cuatro, lo que significa que existen cuatro posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado, (ver Tabla 11).

No.	Camino Básico
1	1-2-3-4-5-6-9-10-11-12-13-14-17
2	1-2-3-4-5-7-8-9-10-11-12-13-14-17
3	1-2-3-4-5-6-9-10-11-12-13-15-16-17
4	1-2-3-4-5-7-8-9-10-11-12-13-15-16-17

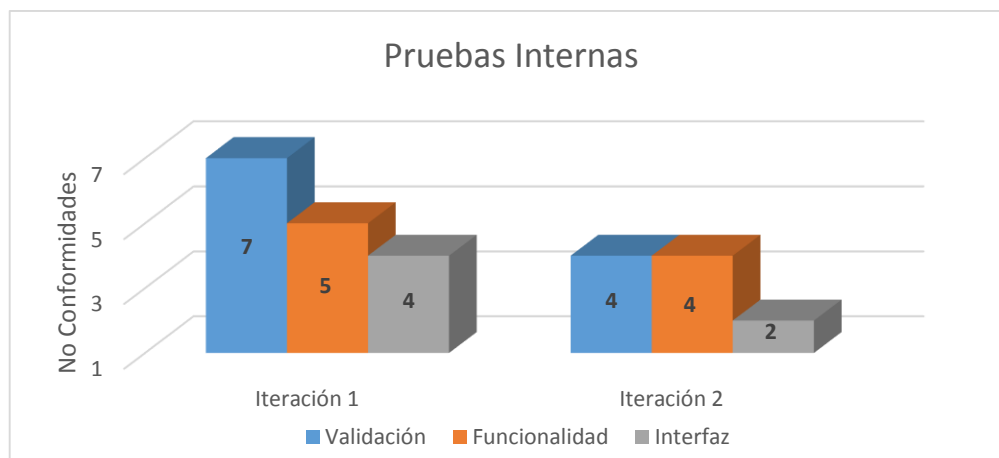
**Tabla 10. Caminos básicos del flujo.**

Con la especificación de los caminos básicos se procedió a ejecutar los casos de pruebas para cada uno de estos. Para definir los casos de pruebas se consideraron las descripciones, condiciones de ejecución, entradas, resultados esperados y los resultados obtenidos. Estos casos de pruebas pueden ser consultados en el documento *Casos de Pruebas*, propuesto por (Durán Bolaño, y otros, 2015), ubicado en la carpeta expediente de proyecto BCGUI.

### Resultados del método de prueba de caja negra

La biblioteca de componentes fue sometida a dos iteraciones de pruebas de caja negra. En la primera iteración se detectaron 16 no conformidades de aplicación. Dentro de las que se encuentran siete de validación, cuatro de interfaz y cinco de funcionalidad. Las no conformidades fueron resueltas posterior a su detección, lo que permitió que la aplicación pasara a una segunda iteración.

En esta nueva iteración se detectaron 10 no conformidades de aplicación. Dentro de las que se encuentran cuatro de funcionalidad, dos de interfaz y cuatro de validación. Estas no conformidades encontradas fueron resueltas al igual que en la iteración anterior obteniéndose resultados satisfactorios.



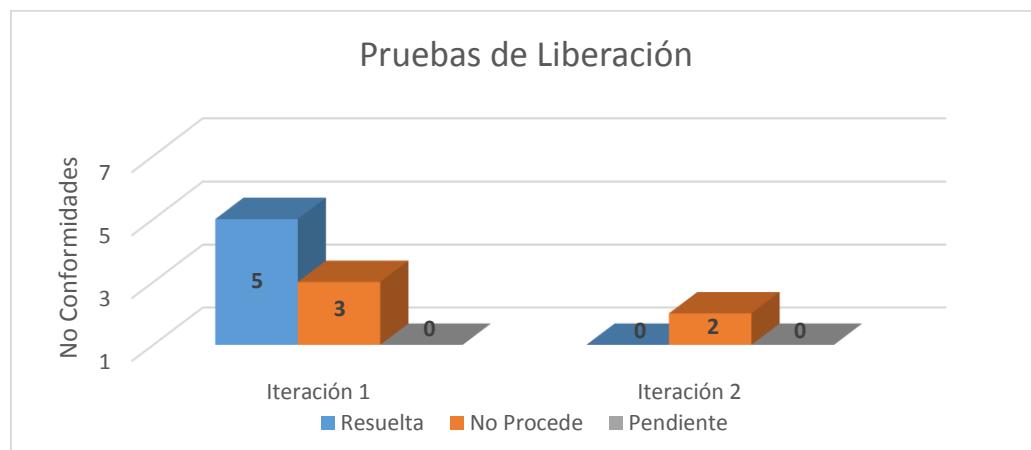
**Imagen 24. Gráfica de los resultados de las pruebas internas.**

El resultado de las pruebas internas permitió detectar no conformidades en el funcionamiento, interfaz y validación de la aplicación, las cuales fueron resueltas por el equipo de desarrollo.

## Resultados de las pruebas de liberación

En las pruebas de liberación, la aplicación fue sometida a dos iteraciones de prueba de caja negra realizadas por el Grupo de calidad de CEGEL. En la primera iteración se detectaron ocho no conformidades de aplicación. De las cuales cinco fueron resueltas satisfactoriamente y tres no procedieron, quedando ninguna pendiente, lo que permitió que la aplicación pasara a una segunda iteración.

En la segunda iteración a la que fue sometida la aplicación se detectaron dos no conformidades, de las cuales ambas no proceden. Por tanto los resultados obtenidos fueron satisfactorios.



**Imagen 25. Gráfica de los resultados de las pruebas de liberación.**

El resultado de las pruebas de liberación permitió verificar la calidad de la aplicación en cuanto a sus funcionalidades. Prueba de lo planteado anteriormente constituye el Acta de Liberación Interna de Productos Software emitida, en la cual consta que el software realizado cumple con las condiciones de calidad necesarias (ver Anexo 5).

### 3.3 Validación mediante caso de estudio

Para las pruebas de aceptación se utilizó un caso de estudio que se le muestra al cliente, para determinar si los componentes están listos para ser implantados, para el uso operativo en el entorno del usuario final. Un caso de estudio ofrece importantes resultados e información que no puede ser encontrada por medio de los métodos cuantitativos y cualitativos tradicionales. Su aplicación posibilita ir observando y comparando los resultados que se esperan alcanzar en el desarrollo de una investigación de una forma real y con resultados medibles (Monge, 2010).

Para el desarrollo del caso de estudio se utilizó una computadora con las siguientes prestaciones:

Tipo de CPU: Intel Celeron Procesador E3400, 1M Cache, 2.60 GHz, 800 MHz.

Memoria del sistema: 2048 MB DDR3-1066 DDR3 SDRAM.

El caso de estudio propone una aplicación de interfaz gráfica que muestra una vista por cada componente implementado para la biblioteca de interfaz gráfica. Las vistas se identifican con el nombre de los componentes con el objetivo de probar sus principales funcionalidades desarrolladas.

Se proponen las siguientes vistas para la evaluación.

1. Funcionalidades del componente de interfaz gráfica JComboboxObjetoFX.
2. Funcionalidades del componente de interfaz gráfica JTablaObjetoFX.
3. Funcionalidades del componente de interfaz gráfica JListaObjetoFX.
4. Funcionalidades del componente de interfaz gráfica JPasswordFieldFX.
5. Funcionalidades del componente de interfaz gráfica JNavigationMenuFX.
6. Funcionalidades del componente de interfaz gráfica JFormateadorFX.
7. Funcionalidades del componente de interfaz gráfica JTextAreaFX.
8. Funcionalidades del componente de interfaz gráfica JIPFX.
9. Funcionalidades del componente de interfaz gráfica JSlidePaneFX.
10. Funcionalidades del componente de interfaz gráfica JCollapsibleMenuFX.
11. Funcionalidades del componente de interfaz gráfica JBannerFX.
12. Funcionalidades del componente de interfaz gráfica JTreeViewFX.

### Vista 1: Funcionalidades del componente de interfaz gráfica JComboboxObjetoFX.

En esta evaluación se tienen en cuenta funcionalidades como la lista de elementos importada por el componente, especificar el atributo de los valores de la lista a mostrar, obtener valor del elemento seleccionado y eliminar elemento seleccionado.

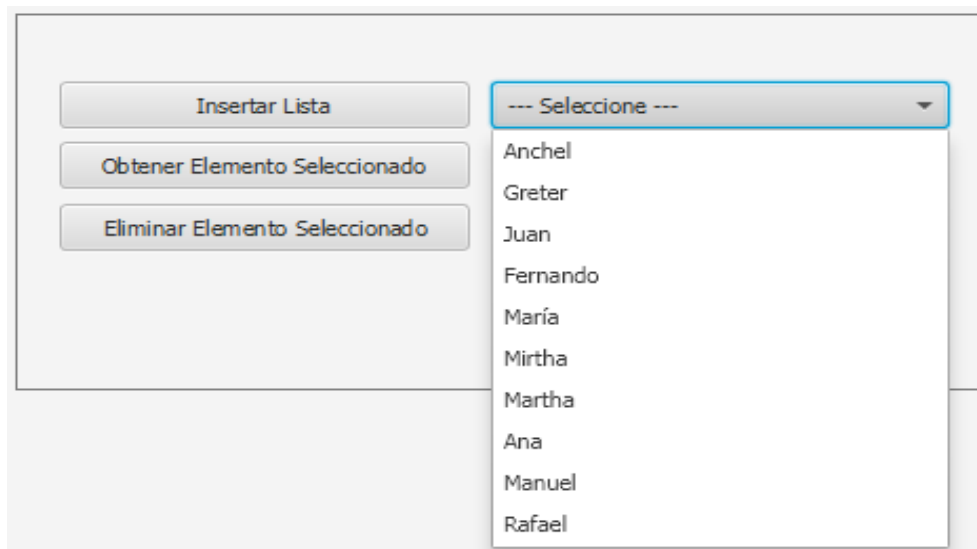


Imagen 26. Vista JComboboxObjetoFX.

### Observación 1:

Se especificó en las propiedades del componente en la herramienta de diseño, el atributo de los elementos que se muestran de la lista y luego se importó la lista, el componente permitió mostrar como resultado los elementos importados de la lista. El usuario pudo interactuar con las funcionalidades obtener valor de elemento seleccionado o eliminar elemento seleccionado en dependencia de su elección.

### Vista 2: Funcionalidades del componente de interfaz gráfica JTablaObjetoFX.

En esta evaluación se tienen en cuenta funcionalidades como la lista de elementos importada por el componente, especificar valores de atributos a mostrar, mostrar nombre de columnas mediante los atributos especificados, mostrar lista de elementos, seleccionar tuplas, obtener valores de tuplas seleccionadas, eliminar tuplas seleccionadas, paginar tabla y filtrar búsqueda de elementos.

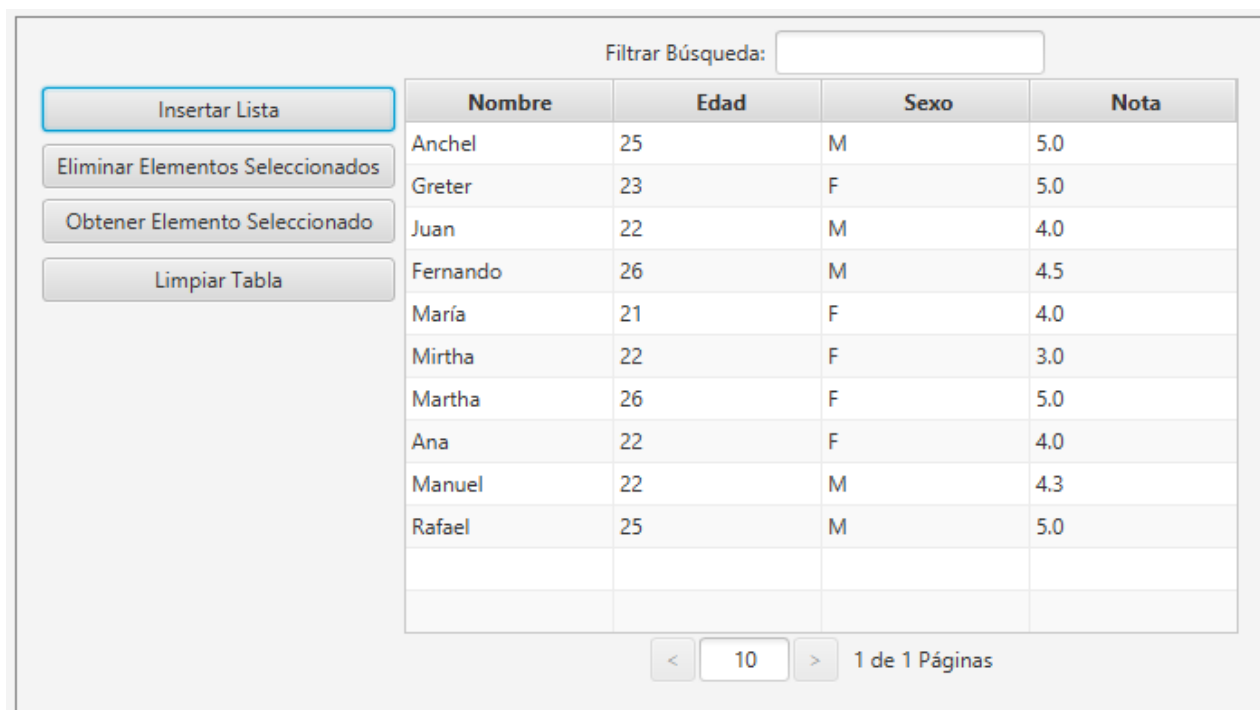


Imagen 27. Vista JTablaObjetoFX.

### Observación 2:

Se especificó en las propiedades del componente en la herramienta de diseño, los atributos de los elementos que se muestran y luego se importó la lista de elementos. El componente permitió mostrar como resultado la conformación de la tabla mediante los nombres de las columnas de los atributos especificados y cada uno de los elementos de la lista importada. El usuario pudo interactuar con las

funcionalidades mostradas para obtener los valores de los elementos, eliminar elementos seleccionados, filtrar búsqueda y paginar la tabla en dependencia de su elección.

### Vista 3: Funcionalidades del componente de interfaz gráfica JListaObjetoFX.

En esta evaluación se tienen en cuenta funcionalidades como la lista de elementos importada por el componente, especificar valores de atributos a mostrar, obtener valor de elemento seleccionado y eliminar elementos seleccionados.

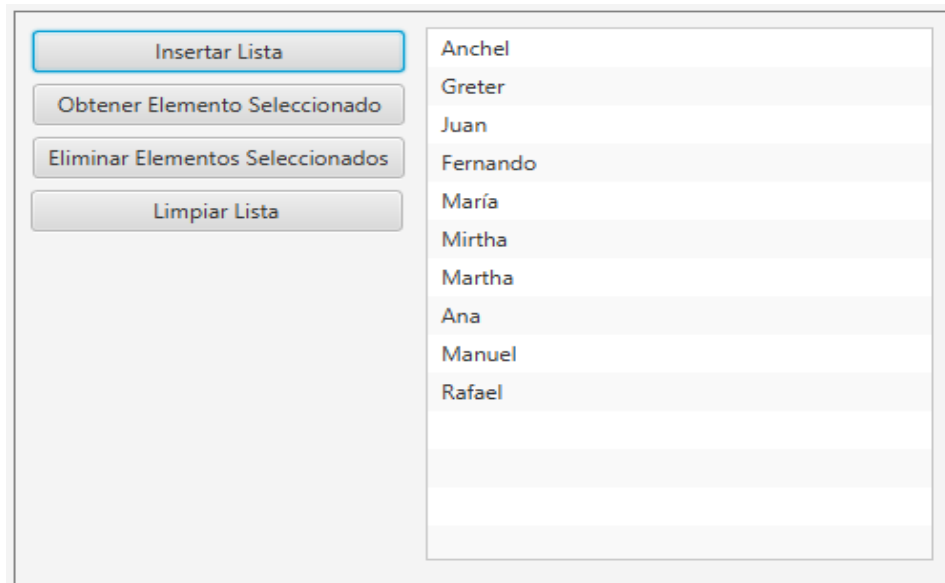


Imagen 28. Vista JListaObjetoFX.

### Observación 3:

Se especificó en las propiedades del componente en la herramienta de diseño, el atributo de los elementos que se muestran de la lista y luego se importó la lista, el componente permitió mostrar los elementos importados. El usuario pudo interactuar con las funcionalidades para obtener el valor del elemento seleccionado o eliminar elementos seleccionados en dependencia de su elección.

### Vista 4: Funcionalidades del componente de interfaz gráfica JPasswordFieldFX.

En esta evaluación se tienen en cuenta funcionalidades como ocultar caracteres, modificar carácter que oculta el texto plano de las contraseñas y mostrar el texto de la contraseña insertada.

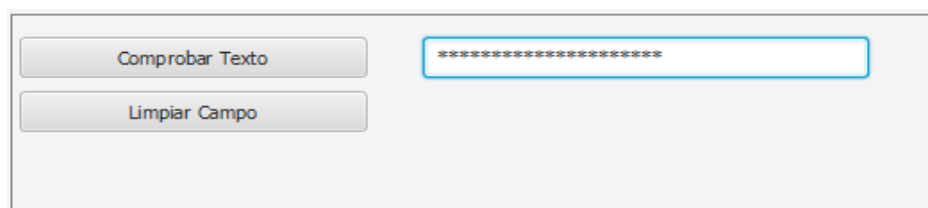


Imagen 29. Vista JPasswordFieldFX.



#### Observación 4:

El componente permitió gestionar el uso de las contraseñas en la aplicación, además, proporciona la funcionalidad cambio de carácter para adaptar la vista del componente en dependencia de como desee mostrarla el desarrollador en su aplicación. Como se puede apreciar en la vista, el componente presenta una cadena de caracteres ocultos mediante el símbolo asterisco.

#### Vista 5: Funcionalidades del componente de interfaz gráfica JNavigationMenuFX.

En esta evaluación se tienen en cuenta funcionalidades como: mostrar la secuencia del camino que sigue un usuario dentro de la aplicación y validar existencia única de ubicación.

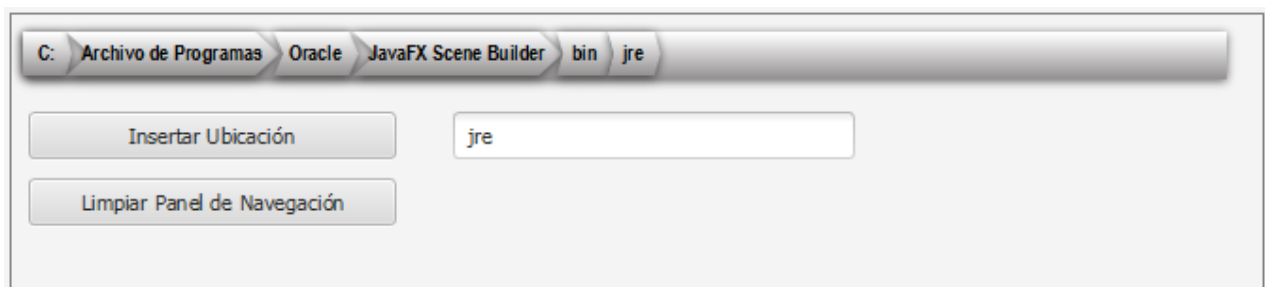


Imagen 30. Vista JNavigationMenuFX.

#### Observación 5:

Mientras el usuario navega dentro de la aplicación, el componente va mostrando una secuencia de botones que van especificando el camino que sigue el usuario. La figura muestra una simulación del funcionamiento, a medida que en la aplicación se accede a un panel, el componente va insertando el nombre del panel para indicar el lugar en que se encuentra. Permitiendo al usuario una interacción dinámica con la aplicación, ya que puede dar saltos entre las diferentes secciones de la aplicación.

#### Vista 6: Funcionalidades del componente de interfaz gráfica JFormateadorFX.

En esta evaluación se tienen en cuenta funcionalidades como validar formato de una cadena de caracteres, restringir entrada de caracteres y selección de expresiones regulares definidas en las propiedades del componente en la herramienta de diseño.

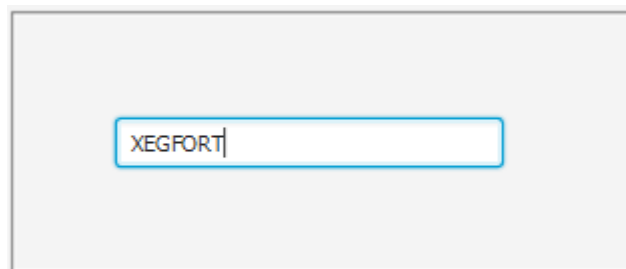


Imagen 31. Vista JFormateadorFX.

### Observación 6:

Se especificaron en las propiedades del componente en la herramienta de diseño, las validaciones necesarias de entradas de caracteres al campo de texto, estas propiedades restringen además, la entrada de combinaciones de palabras formadas por letras; números enteros; letras y números enteros; iniciales de nombres, entre otros. En la vista asociada al componente JFormateadorFX en el caso de estudio, se restringió la entrada de iniciales de nombre.

### Vista 7: Funcionalidades del componente de interfaz gráfica JTextAreaFX.

En esta evaluación se tienen en cuenta funcionalidades como almacenar cantidades de textos, gestionar textos insertados, ajustar líneas de texto, validar entrada de caracteres en tiempo de ejecución, cambiar estilo del formato de texto normal, negrita o cursiva en tiempo de ejecución.

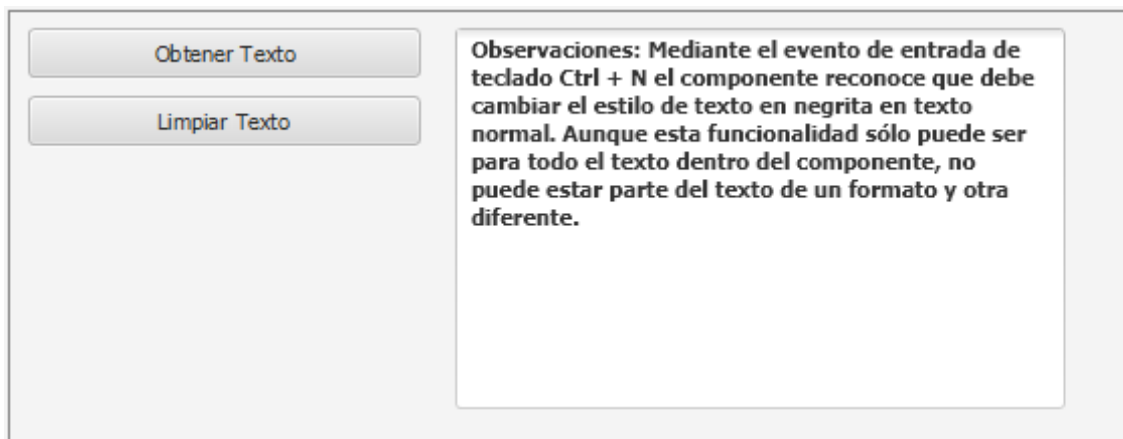


Imagen 32. Vista JTextAreaFX.

### Observación 7:

Se especificó en las propiedades del componente que no valide ninguna restricción de entrada de caracteres. El usuario puede insertar cantidades de textos para la descripción de temas. Con el uso de las combinaciones de eventos de teclado el componente pudo cambiar el formato del texto para detallar las descripciones, obteniendo un avanzado componente de área de texto para el desarrollo de interfaces visuales.

### Vista 8: Funcionalidades del componente de interfaz gráfica JIPFX.

En esta evaluación se tienen en cuenta funcionalidades como registrar direcciones de IP, restringir entrada de números enteros entre 0 y 255 y obtener la dirección IP registrada.

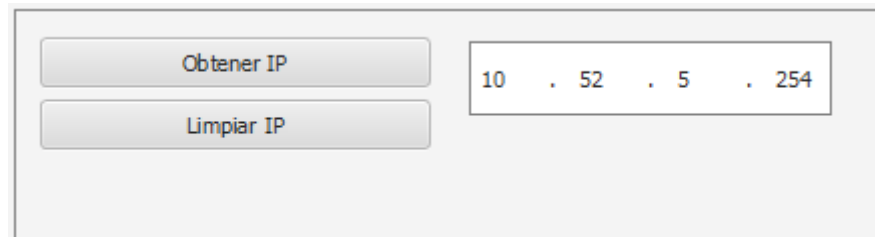


Imagen 33. Vista JIPFX.

### Observación 8:

El componente permitió registrar direcciones de IP para indicarle a la aplicación que su rango de trabajo se encuentra dentro de la subred indicada. Presenta funcionalidades avanzadas que validan la entrada de números enteros para el uso de direcciones IP.

### Vista 9: Funcionalidades del componente de interfaz gráfica JSlidePaneFX.

En esta evaluación se tienen en cuenta funcionalidades como deslizar secuencia de paneles hacia delante y hacia atrás, gestionar botones de navegación de los flujos de paneles y habilitar o deshabilitar los botones de navegación.

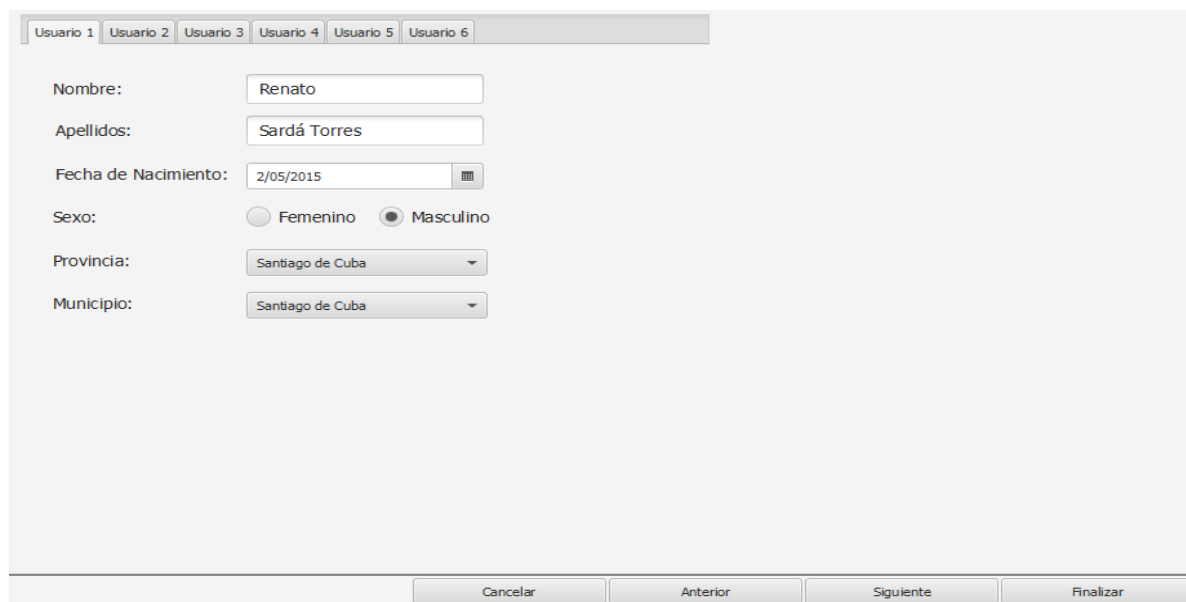


Imagen 34. Vista JSlidePaneFX.

### Observación 9:

Permitió a los usuarios realizar una simulación de navegación entre interfaces y paneles en el caso de estudio. Los botones de navegación deslizan los paneles que muestran la información suavemente proporcionando una agradable manipulación de la interfaces visuales. Los desarrolladores pueden gestionar los botones del panel wizard en dependencia de las funcionalidades que necesiten, así como Cancelar, Anterior, Siguiente o Finalizar, entre otros.

### Vista 10: Funcionalidades del componente de interfaz gráfica JCollapsibleMenuFX.

En esta evaluación se tienen en cuenta funcionalidades como plegar menú, desplegar menú y mostrar funcionalidades de los flujos de trabajo.

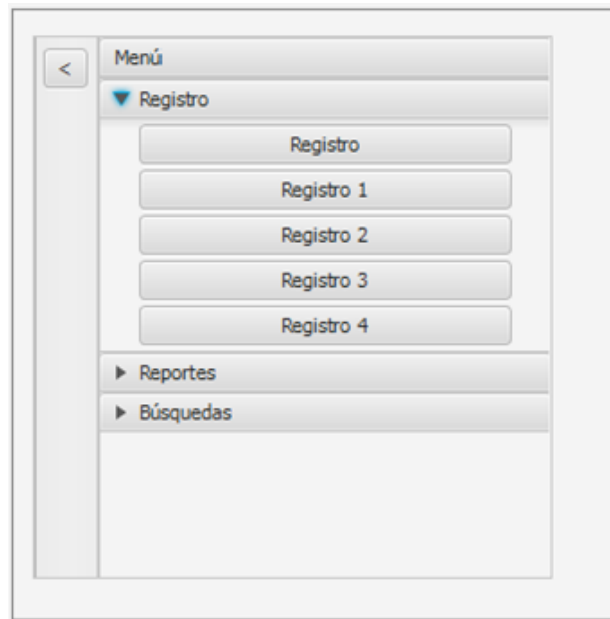


Imagen 35. Vista JCollapsibleMenuFX.

### Observación 10:

Las funcionalidades del componente permitieron a los desarrolladores insertarle un menú a la aplicación que muestre las secciones de la aplicación. Con las funcionalidades de plegar y desplegar el menú, los usuarios pudieron manipular la interfaz de la aplicación permitiéndoles agrandar el área de trabajo. El componente presenta una propiedad capaz de modificar el tiempo de animación del menú, que permitió observar la velocidad del efecto como aumentaba o disminuía, esta propiedad se deja a consideración del desarrollador para su utilización.

### Vista 11: Funcionalidades del componente de interfaz gráfica JBannerFX.

En esta evaluación se tienen en cuenta funcionalidades como mostrar cartel (banner) para aplicaciones de escritorio, conservar aspecto original de la imagen y modificar tamaño de la imagen sin distorsión.



Imagen 36. Vista JBannerFX.

### Observación 11:

La vista permitió simular un componente que importa una imagen como cartel o banner en la aplicación. Dentro de sus funcionalidades se encuentran propiedades como juego de CSS para el estilo de las imágenes. Estas permitieron conservar el aspecto original para mantener la calidad en altas resoluciones de pantallas, juego de bordes, insertar varias imágenes, entre otras funcionalidades avanzadas para el uso del componente en la aplicación.

### Vista 12: Funcionalidades del componente de interfaz gráfica JTreeViewFX.

En esta evaluación se tienen en cuenta funcionalidades como la lista de elementos importada por el componente, especificar valor a mostrar de nodo raíz, especificar valor a mostrar de atributo de agrupación, obtener valor de elemento seleccionado y eliminar elementos seleccionados.

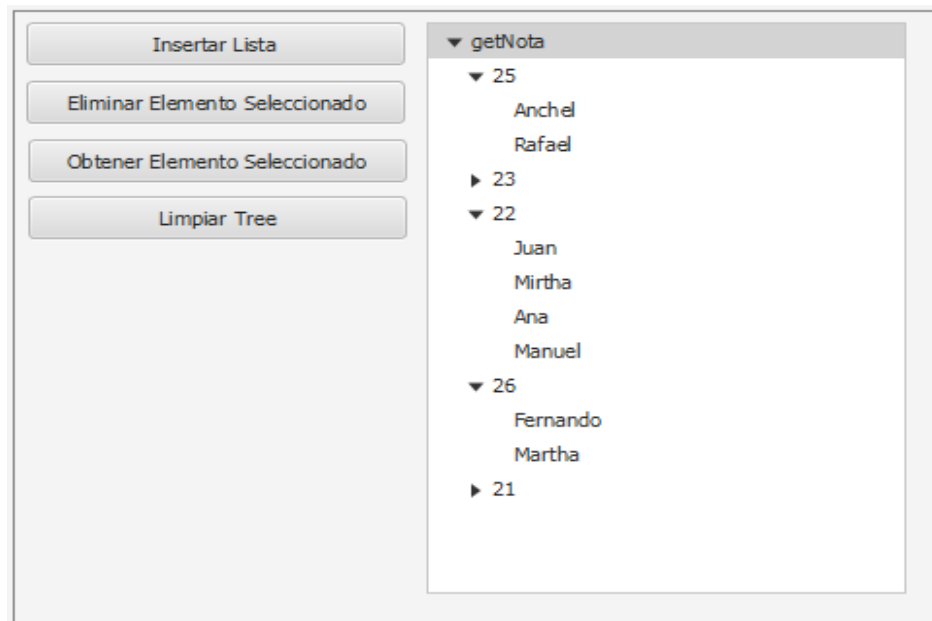


Imagen 37. Vista JTreeViewFX.

### Observación 12:

Se especificó en las propiedades del componente en la herramienta de diseño, el atributo de los elementos a mostrar de la lista así como el atributo nodo raíz, atributo de agrupación y luego se importó la lista de elementos, el componente mostró como resultado los elementos importados de la lista organizados en forma de árbol o por jerarquía. El usuario pudo interactuar con las funcionalidades obtener valor de elemento seleccionado o eliminar elementos seleccionados en dependencia de su elección.

## Análisis de los resultados

La biblioteca de componentes de interfaz gráfica para aplicaciones de escritorio desarrolladas en Java presenta para su utilización en el desarrollo de interfaces visuales, 12 componentes gráficos desarrollados con la tecnología JavaFX. Estos componentes presentan diferentes propiedades y funcionalidades avanzadas para el trabajo con listas, textos, menú, paneles, árboles, imágenes, direcciones IP, entre otros; que disminuyen la complejidad en el desarrollo de interfaces visuales. Durante el desarrollo y validación de las funcionalidades del caso de estudio, se pudo apreciar la calidad de los componentes en la creación de las interfaces. El cliente se mostró satisfecho con el producto entregado, prueba de lo planteado anteriormente constituye el Acta de Aceptación del cliente (Ver Anexo 6).

### 3.4 Validación de las variables de investigación

Para validar las variables de investigación, se realizó un análisis de las funcionalidades de los anteriores componentes de la biblioteca de interfaz gráfica de XEGFORT y los nuevos componentes de la biblioteca implementada en JavaFX.

Componentes	Antes	Después
JFormateadorFX	El formateador presentaba deficiencias en las expresiones regulares definidas para validar direcciones de correos electrónicos, URL, números de carné de identidad, entre otros. Además no cuenta con una selección de expresiones regulares, por lo que se necesitaba implementaciones adicionales.	Permite manejar la inserción de caracteres al componente para restringir la entrada de cadenas de caracteres formadas por letras; números enteros; letras y números enteros; iniciales de nombres; nombres, entre otros. Cuenta con una lista de expresiones regulares para realizar las validaciones de cadenas de caracteres, facilitando a los desarrolladores no tener que implementar validaciones adicionales, permitiendo disminuir el tiempo de desarrollo de las interfaces visuales.
JComboBoxObjetoFX	No presenta interacción entre la vista y el modelo del componente, provocando incorrecta sincronización de	El componente permite importar una lista de elementos y definir qué atributos mostrar, sin tener que recurrir a varias implementaciones para mostrar la lista de los elementos.

	<p>los datos mostrados, lo que influye en implementaciones y validaciones adicionales para su correcto funcionamiento.</p>	<p>Presenta una correcta interacción entre la vista y el modelo, funcionando de forma adecuada para su utilización. Además contiene funcionalidades como eliminación de elementos y captura de datos, facilitando el trabajo de los desarrolladores al no tener que realizar implementaciones adicionales,</p>
JTablaObjetoFX	<p>No cuenta con funcionalidades necesarias para su uso como filtrado de búsquedas de elementos y paginado de la tabla para mostrar elementos organizadamente en caso de que exista gran cantidad de información. Además, no actualiza la información contenida en la vista, cuando ocurren eventos de eliminación y modificación de los datos.</p>	<p>Presenta las funcionalidades filtrado de búsqueda que permite buscar de forma rápida en todos los campos del componente los elementos similares al criterio de búsqueda. Cuenta con la funcionalidad paginar tabla para mostrar la cantidad de tuplas que indique el usuario y el número de páginas. Presenta la funcionalidad selección múltiple de elementos en caso que el desarrollador necesite interactuar con varios elementos seleccionados, así como eliminación de varios elementos. Presenta sincronización entre la vista y el modelo del componente permitiendo una interacción dinámica de los eventos realizados por el usuario. Permite ocultar las funcionalidades filtrado de búsqueda y paginado en dependencia de la elección del desarrollador. Cuenta con funcionalidades y propiedades avanzadas, que disminuyen el tiempo de desarrollo de las interfaces, facilitando al desarrollador no tener que implementarlas.</p>
JCollapsibleMenuFX	<p>No presenta funcionalidades para ampliar el área de trabajo, además no organiza por jerarquía los sub-elementos contenidos en su</p>	<p>El componente cuenta con las funcionalidades plegar y desplegar el menú, permitiendo a los desarrolladores no tener que buscar soluciones adicionales para ampliar el área de trabajo. Presenta las</p>

	interior que permiten indicar las ubicaciones de las áreas de trabajo de XEGFORT.	herramientas para organizar los sub-elementos contenidos en su interior. Cuenta con una barra horizontal en la parte izquierda que permite adicionar otros eventos de estado al menú, permitiendo no tener que emplear tiempo adicional en su desarrollo.
JIPFX	El componente no presenta deficiencias.	Aunque no influye en la disminución del tiempo, permite sustituir el componente anterior, puesto que presenta todas las validaciones necesarias para su funcionamiento. Además, por ser desarrollado con la tecnología JavaFX y su integración con la herramienta de diseño, para su utilización en las interfaces visuales.
JTextAreaFX	Los desarrolladores necesitan realizar funcionalidades adicionales para que el texto se ajuste a las dimensiones del componente. No presenta implementaciones para cambiar el formato de las letras del texto en tiempo de ejecución.	Agrega las funcionalidades para cambiar el formato del texto insertado en el componente en tiempo de ejecución, en letras negritas o cursivas. Permite ajustar el texto insertado para acomodarlo a las dimensiones del componente. Permite manejar la inserción de caracteres al componente para restringir la entrada de cadenas de caracteres formadas por letras; números enteros; letras y números; textos en mayúsculas, entre otros. Cuenta con una lista de expresiones regulares para realizar las validaciones de cadenas de caracteres, facilitando a los desarrolladores no tener que implementar validaciones adicionales, permitiendo disminuir el tiempo de desarrollo de las interfaces visuales.
JBannerFX	No presenta funcionalidades para importar imágenes en caso de ser necesario	El componente presenta la funcionalidad, para importar diferentes imágenes de ser necesario, facilitándole al desarrollador no



	<p>cambiarlas, sin tener que modificar el código del componente. No tiene propiedades para el uso de CSS.</p>	<p>tener que implementar estas operaciones. Presenta un mejor algoritmo de filtrado de calidad o más rápido al transformar o escalar la imagen de origen para que se ajuste a las dimensiones del componente. Permite utilizar CSS para dar mejor calidad visual a la vista, permitiendo a los desarrolladores aprovechar sus ventajas sin tener que emplear tiempo adicional en el desarrollo de funcionalidades para CSS.</p>
JListaObjetoFX	<p>No presenta interacción entre la vista y el modelo del componente, influyendo en pérdida de tiempo en la realización de métodos y operaciones adicionales para validar el correcto funcionamiento.</p>	<p>Presenta la funcionalidad selección múltiple de elementos en caso que el desarrollador necesite interactuar con varios elementos seleccionados. El componente permite importar una lista de elementos y definir qué atributos de la lista mostrar en la vista, sin tener que recurrir a varias implementaciones para mostrar la lista de los elementos. Presenta una correcta interacción entre la vista y el modelo, funcionando de forma adecuada para su utilización. Además contiene funcionalidades cómo eliminación de elementos y captura de datos, facilitando el trabajo de los desarrolladores al no tener que realizar implementaciones adicionales,</p>
JPasswordFieldFX	<p>No tiene funcionalidad de selección para cambiar los caracteres que oculta la contraseña insertada.</p>	<p>Presenta validaciones para seleccionar diferentes tipos de caracteres que ocultan la contraseña insertada en el campo de texto. Aunque no influye en la disminución del tiempo, permite incorporarse en la biblioteca de interfaz gráfica implementada, por ser desarrollado con la tecnología JavaFX y su integración con la herramienta de diseño, para su utilización en las interfaces visuales.</p>

JNavigationMenuFX	Para mostrar el camino trazado se apoyaban de validaciones y funcionalidades adicionales para mostrar la secuencia de flujos en el marco de trabajo.	El componente muestra la secuencia del camino en la aplicación, auxiliándose de las funcionalidades y validaciones necesarias para definir el camino seguido por los usuarios. Permite validar la inserción múltiple de las ubicaciones. Presenta una interacción dinámica entre el usuario y la aplicación mediante los eventos del mouse para navegar en la aplicación al alcance de un clic. Cuenta con un juego de CSS para mejorar la calidad visual. Este contribuye a disminuir el tiempo de desarrollo.
JSlidePaneFX	Se realizaba exceso de implementaciones para deslizar las interfaces visuales. Validando funcionalidades para habilitar o deshabilitar los botones que manejan el flujo de los paneles. Invirtiendo para ello, gran cantidad de tiempo en realizar estas operaciones para mantener una secuencia dinámica.	Permite importar una lista de interfaces para deslizarlas como una secuencia de paneles o vistas que muestran la información de la aplicación a los usuarios. Permite gestionar los botones de navegación de la aplicación que inicializan, cambian o finalizan los flujos de trabajo. Los desarrolladores no necesitan estar al tanto de las interfaces, ni de la visibilidad de los botones que manejan el flujo de la secuencia de los paneles. Debido, que el componente se encarga de manipular estos eventos dinámicamente, permitiendo a los desarrolladores no tener que realizar estas funcionalidades.
JTreeViewFX	Utilizaban el componente original de Swing, sobre el cual realizaban todas las funcionalidades y validaciones necesarias para su utilización. Para ello, empleaban tiempo adicional en poder adaptar el	El componente permite importar y mostrar una lista de elementos por jerarquía en tres niveles, facilitando la estructura para mostrar elementos en forma jerárquica. Permite definir en sus propiedades los atributos a mostrar en cada uno de los niveles del componente, proporcionando a los desarrolladores no tener que implementar

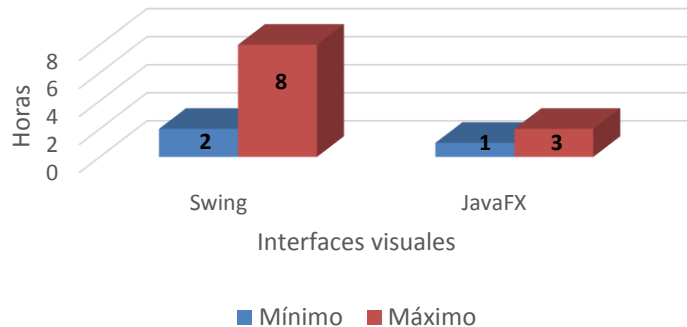
	<p>funcionamiento del componente a sus necesidades.</p>	<p>estas funcionalidades. Presenta la funcionalidad selección múltiple de elementos en caso que el desarrollador necesite interactuar con varios elementos seleccionados. Estas funcionalidades permiten personalizar el componente de acuerdo a las necesidades del negocio. Además, disminuye el tiempo empleado en realizar funcionalidades adicionales para utilizar el componente.</p>
--	---	---

**Tabla 11. Validación de las variables de investigación.**

El análisis realizado anteriormente, permite demostrar que los componentes de la biblioteca de interfaz gráfica en JavaFX, disminuyen el tiempo de desarrollo de las interfaces visuales. De esta forma se constató que las funcionalidades y propiedades de los componentes utilizando la tecnología JavaFX, facilitan su utilización a los especialistas de XEGFORT para desarrollar interfaces visuales para aplicaciones en Java.

Mediante la entrevista realizada a los especialistas de CEGEL que interactúan con el marco de trabajo XEGFORT en el desarrollo de aplicaciones, se obtuvieron resultados que demuestran que el tiempo invertido en el desarrollo de las interfaces visuales se encuentra en intervalos de dos a ocho horas. Por otro lado, un especialista con experiencia en XEGFORT simuló el desarrollo de varias interfaces complejas utilizando gran cantidad de componentes. Lo que demuestra que en el diseño de las interfaces en Swing, se dedica tiempo adicional para crear y validar las funcionalidades de las acciones que contienen el código de las vistas. Trayendo consigo un aumento de dos a cuatro horas adicionales en conformar e implementar una interfaz que normalmente emplearía hasta cuatro horas para su desarrollo.

## Tiempo de desarrollo de las interfaces visuales



**Imagen 38. Gráfica de los tiempos de desarrollo de las interfaces visuales.**

Para demostrar la disminución del tiempo de desarrollo de las interfaces visuales aprovechando las funcionalidades de la biblioteca de componentes en JavaFX, el especialista en XEGFORT simuló un ejemplo similar al de las interfaces realizadas en Swing. Se emplea para ello, los componentes de la herramienta de JavaFX Scene Builder y los de la biblioteca de interfaz gráfica desarrollados en este trabajo.

Al utilizar la herramienta de diseño JavaFX Scene Builder, el especialista demostró con la creación e implementación de las funcionalidades de las interfaces, que invierte un tiempo de hasta tres horas. Pues el manejo de las propiedades y operaciones de los componentes se ajustan de acuerdo a las necesidades, debido que el especialista no necesita implementar funcionalidades innecesarias, ni validaciones que influyan en tiempo adicional como sucede en las interfaces de Swing. Demostrándose la viabilidad que trae el uso de la biblioteca de interfaz gráfica desarrollada, en la reducción del tiempo de desarrollo de las interfaces visuales de las aplicaciones que utilizan el marco de trabajo XEGFORT.

## Conclusiones Parciales

- ✓ Se validaron las disciplinas de desarrollo abarcadas durante el desarrollo del producto como buenas prácticas permitiendo obtener un producto de calidad.
- ✓ Las técnicas y métricas permitieron validar los requisitos de la aplicación y el diseño de la solución.
- ✓ La verificación de la implementación, permitió realizar las pruebas de software a la biblioteca de componentes mediante las disciplinas de pruebas que propone la metodología, obteniendo resultados satisfactorios.
- ✓ La validación mediante caso de estudio permitió evaluar el método propuesto a partir de los resultados obtenidos mediante su ejecución en una interfaz de prueba.
- ✓ Las observaciones realizadas en el desarrollo de cada vista del caso de estudio constataron su utilidad en la solución.
- ✓ Se realizó la validación de las variables de investigación, donde se mostró un análisis de las actuales funcionalidades de los componentes de interfaz gráfica de la biblioteca propuesta como solución, que influyen en la disminución del tiempo de desarrollo de las interfaces visuales.

## CONCLUSIONES GENERALES

Con la culminación del presente trabajo de diploma se puede concluir que se desarrollaron todas las tareas para cumplir con el objetivo propuesto, destacando que:

- ✓ La realización del marco teórico conceptual de la investigación, permitió detectar las deficiencias existentes y la incorporación de nuevas funcionalidades para disminuir el tiempo de desarrollo de las interfaces visuales de las aplicaciones desarrolladas en Java.
- ✓ Se implementó la biblioteca de componentes gráficos para el desarrollo de interfaces visuales en XEGFORT, que contribuyó a disminuir las deficiencias existentes utilizando las herramientas y tecnologías definidas.
- ✓ Las pruebas realizadas arrojaron resultados satisfactorios, demostrando la disminución del tiempo de desarrollo de las interfaces visuales y se constató la calidad de la biblioteca de componentes de interfaz gráfica desarrollada, utilizando para ello técnicas y métricas aplicables al desarrollo de la solución.

## RECOMENDACIONES

Se recomienda:

- ✓ Utilizar la biblioteca propuesta en otros centros de desarrollo de la UCI, para promover el trabajo con la tecnología JavaFX.
- ✓ Adaptar el marco de trabajo XEGFORT, para la utilización de la biblioteca de componentes en la realización de interfaces gráficas.
- ✓ Mejorar la biblioteca de componentes de acuerdo a la retroalimentación de los programadores que desarrollan aplicaciones utilizándola.

## BIBLIOGRAFÍA

- Álvarez, Ramiro Perez. 2012.** T7A OPERACIONALIZACIÓN DE VARIABLES. *T7A OPERACIONALIZACIÓN DE VARIABLES*. [En línea] 2012. <http://metinvc.blogspot.com/2012/02/t6a-variables-recurso-analitico.html>.
- Bautista. 2012.** 2012.
- Berndtsson, Mikael. 2008.** *ThesisProjects Pittsburgh: Springer*. 2008. 978-1-84800-008-7.
- Beyris Soulyar, Liliam Celia, y otros. 2010.** *Proceso de medición y análisis para el polo de hardware y automática*. 2010.
- CALISOFT. 2011.** *Aseguramiento de la Calidad del proceso y el producto*. La Habana : s.n., 2011.
- CEGEL, Centro de Gobierno Electrónico. 2006.** Suite de Gestión de Proyecto. [En línea] 2006. [Citado el: 6 de Febrero de 2015.] <http://gespro.cegel.prod.uci.cu/>.
- Christensson, Per. 2015.** Tech Terms.com. [En línea] 2015. <http://techterms.com>.
- Claro Sánchez, y otros. 2011.** 2011.
- de la Torre Llorente, César. 2010.** Guía de Arquitectura N-Capas orientada al Dominio. 2010.
- Diseño, Metricas de. 2015.** Scribd. [En línea] 2015. <https://es.scribd.com/doc/170276664/Metricas-de-diseno>.
- Diseño, Patrones de. 2014.** Patrones de Diseño. [En línea] 29 de Enero de 2014. <https://eseida.wikispaces.com/file/view/Tema+6+-+Patrones+de+Dise%C3%B1o.pdf>.
- Durán Bolaño, Anchel y Gómez Martínez, Greter. 2015.** Biblioteca de componentes gráficos de interfaz de usuario para aplicaciones de escritorio desarrolladas en java. La Habana : s.n., 2015.
- Fernández Romero, Yenisleidy y Díaz González, Yanette . 2012.** Patrón Modelo-Vista-Controlador. [En línea] 2012. <http://revistatelematica.cujae.edu.cu/index.php/tele/article/viewFile/15/10>.
- Frías, Roberley Cuadra. 2012.** *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*. La Habana, : s.n., 2012. pág. 23, Tesis.
- . 2012. *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*. La Habana : s.n., 2012. pág. 21, Tesis.
- . 2012. *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*. La Habana : s.n., 2012. pág. 22, Tesis.
- FXML, Introduction to. 2014.** Oracle. *Oracle*. [En línea] 2014. [http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction\\_to\\_fxml.html](http://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html).
- Gamma, Erich, y otros. 2005.** Microsoft Developer Network. [En línea] 2005. <http://msdn.microsoft.com/>.
- Hamilton, Graham. December 1997.** *JavaBeans*. s.l. : Sun Microsystems Inc., December 1997.
- Holzner, Steve. 2006.** *Design Patterns For Dummies*. 2006.
- Humberto Cervantes. 2010.** SG. SG. [En línea] ABRIL de 2010. [Citado el: 22 de nOVIEMBRE de 2014.] [sg.com.mx/revista/27/arquitectura-software](http://sg.com.mx/revista/27/arquitectura-software).
- Institute, Project Management. 2012.** Project Management Institute. [En línea] 2012. [Citado el: 7 de Febrero de 2015.] <http://www.liderdeproyecto.com/sitio/component/agorapro/default-topic/1299-diferencia-entre-la-definici%C3%B3n-de-alcance-del-producto-y-del-proye>.
- ISO. 2006.** *Determinación de los Requerimientos de Calidad del Producto Software basados en Normas Internacionales*. 2006.
- Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000.** *El Proceso Unificado de Desarrollo de Software*. Madrid : Addison Wesley, 2000. pág. 464.
- JavaFX, Arquitectura. 2013.** Oracle . [En línea] 2013. [Citado el: 8 de Febrero de 2015.] <https://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm>.
- Jorrín Perdomo, Jorge Yuniel. 2013.** *Proyecto Técnico SIRECC*. Universidad de las Ciencias Informáticas (UCI). La Habana : s.n., 2013, 2013. Vol. Técnico.
- Juristo, Natalia, Moreno, Ana M. y Vegas, Sira . 2006.** *Técnicas de evaluación de software*. 2006.

**Kruchten. 2000.** 2000.

**Liliam Celia Beyris Soulayr, Ludisley La Torre Hernández, Mariela Cepero Núñez, Annelis, Irina Marrero Borges, Yailién Hernández Alba, Katia Onelia Carralero y Amado Espinosa Hidalgo. 2010.** *Proceso de medición y análisis para el polo de hardware y automática.* s.l. : Grupo Editorial Ediciones Futuro ,Serie Científica de la Universidad de las Ciencias Informáticas (SC-UCI) , 2010.

**Mejora, Programa De. 2013.** *Metodología de desarrollo para la actividad productiva de la UCI.* La Habana : Universidad de las Ciencias Informáticas, 2013.

**Monge, E.C. 2010.** *El estudio de casos como metodología de investigación y su importancia en la dirección y administración de empresas.* *Revista Nacional de Administración*, vol. 1, no. 2, pp. 31–54. 2010.

**Oracle Corporation. 2013.** Oracle Corporation. [En línea] 2013. <http://www.oracle.com>.

**Palma, Julio Amado López. 2011.** *Biblioteca de componentes de interfaz de usuario para el.* La Habana : s.n., 2011. pág. 8, Tesis.

**Peláez, Juan. 2009.** *Arquitectura basada en Componentes.* *Arquitectura basada en Componentes.* [En línea] 15 de abril de 2009. <http://www.juanpelaez.com/geek-stuff/arquitectura/arquitectura-basada-en-componentes/>.

**Presman, Roger. 2005.** *Ingeniería del software: el enfoque de un profesional.* 2005.

**Pressman. 2010.** *Ingeniería del Software.* 2010.

—. 2007. *Ingeniería del Software, Un enfoque práctico.* 2007.

**Pressman, Roger. 2005.** *Ingeniería del software: el enfoque de un profesional.* 2005.

—. 2005. Métricas del producto para el software 6ta edición Cap15. *Ingeniería del software: Un Enfoque Practico.* 2005.

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

**Producción, Dirección Técnica de la. 2011.** *Estándar de codificación para Java.* La Habana : s.n., 2011.

**Prototipos, Modelo de Construcción de. 2015.** Scribd. [En línea] 2 de junio de 2015. <http://es.scribd.com/doc/43653369/Modelo-de-Construccion-de-Prototipos#scribd>.

**Raghavan, Sridhar, Zelesnik, Gregory y Ford, Gary A. 1994.** *Lecture Notes on Requirements Elicitation.* Pittsburgh (E.E.U.U.) : Institute (Carnegie Mellon University), 1994.

**Sáez, Ariel Alexis Fierro. 2010.** *Tópicos Avanzados de Programación.* 2010.

**Sanchez, Tamara Rodriguez. 2015.** *Metodología de desarrollo para la actividad productiva de la UCI.* 2015.

**Software, Ingeniería de. 2015.** *Ingeniería de Software.* [En línea] 2015. [Citado el: 7 de Febrero de 2015.] [http://ingenieriadesoftware.mex.tl/63758\\_AUP.html](http://ingenieriadesoftware.mex.tl/63758_AUP.html).

**Software, Ingeniería de. 2003.** Universidad Nacional Abierta y a Distancia. [En línea] 2003.

**Sommerville. 2005.** *Ingeniería del Software 7ma edición.* Madrid : PEARSON EDUCATION, S.A, 2005.

**Sommerville, IAM. 2007.** *Ingeniería del Software 8va edición.* s.l. : Pearson, 2007.

**SublimeSolutions. 2010.** SublimeSolutions. [En línea] 2010. [http://www.proyectosespeciales.com/disenio\\_interfaces\\_graficas/](http://www.proyectosespeciales.com/disenio_interfaces_graficas/).

**Szyperski, Clemens. 2002.** *Component Software: Beyond Object-oriented Programming.* 2002.

**Terreros, Julio Casal. 2005.** Microsoft Developer Network. [En línea] 2005. <http://msdn.microsoft.com/es-es/library/bb972268.aspx#mainSection>.

**UCI, Universidad de las Ciencias Informáticas. 2012.** Universidad de las Ciencias Informáticas. [En línea] 2012. [Citado el: 4 de Febrero de 2015.] <http://www.uci.cu>.

**Visual Paradigm. 2013.** Visual Paradigm. [En línea] 13 de enero de 2013. <http://www.visual-paradigm.com/>.

**Zaldivar, Luis Angel López. 2012.** *Propuesta de solución a la Vista de Arquitectura de Datos del Sistema GESPRO 12.05.* 2012.

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