Universidad de las Ciencias Informáticas
Facultad 5

Título: "*Editor para la Configuración de la Interfaz Gráfica de Ogre3D*"

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Angel Ulise Tabares González

Tutor: M.Sc. Leoder Alemañy Socarrás

Consultante: Ing. Belkis Grissel González Rodríguez

Ciudad de la Habana 2012
"Año 53 de la Revolución Cubana"

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Angel Ulise Tabares González

_____
Firma del Autor

M.Sc. Leoder Alemañy Socarrás

_____
Firma del Tutor

DATOS DE CONTACTO

AUTOR:

Angel Ulise Tabares González.
Universidad de las Ciencias Informáticas,
Ciudad de la Habana, Cuba
E-mail: autabares@estudiantes.uci.cu.

TUTOR:

M.Sc. Leoder Alemañy Socarrás.
Universidad de las Ciencias
Ciudad de la Habana, Cuba
E-mail: lalemany@uci.cu.

DEDICATORIA

A Pedro, quien pronto tendrá un ingeniero en la familia.

A Nereyda, Mismary y Ulise, mis dos madres y mi padre.

A la no tan pequeña Rachel y a Leo.

AGRADECIMIENTOS

A mi familia por la preocupación diaria.

A mi tutor M.Sc. Leoder Alemañy Socarrás, por el apoyo brindado e inestimable empuje desde los primeros momentos.

A quien me animó más que nadie y siempre confió, Midiala.

A cada uno de los profesores de los cuales me nutrí durante esta carrera en las aulas o en el proyecto, sin sus enseñanzas este trabajo no habría ni siquiera comenzado.

A mis amigos Ernesto, Diana, a los viejos del pre, por la ayuda y apoyo de siempre.

A mis compañeros de tesis Alejandro, Jandy, Iván, Juan Carlos y José Andrés por el apoyo brindado durante todo este tiempo.

A todos los que de una manera u otra han contribuido a la feliz culminación de este trabajo.

Y a la Revolución cubana, que me tomó de pequeño de la mano y me ha traído hasta las puertas de la ingeniería.

A todos gracias.

RESUMEN

El proyecto Prolavi del Centro de Informática Industrial de la Universidad de las Ciencias Informáticas necesita desarrollar Laboratorios Virtuales en 3D; para esto utiliza como motor de *render* Ogre3D. Como biblioteca de interfaz gráfica de usuario se utiliza una versión de la que posee el propio motor, que contiene varios componentes formados por *overlays* de Ogre3D; dicha versión fue desarrollada en este mismo proyecto.

Ante los inconvenientes que tiene construir las interfaces gráficas de usuario en tiempo de implementación, porque implica recompilar el código incluso para pequeños cambios, se necesita una herramienta que permita realizar de otro modo esta tarea. El objetivo de este trabajo es diseñar e implementar un software que funcione como diseñador de interfaces gráficas de usuario para la biblioteca usada por el proyecto Prolavi.

Para llevar a cabo este trabajo se ha hecho uso de métodos teóricos tales como el analítico-sintético, histórico-lógico y la modelación; y métodos empíricos como la consulta de fuentes de información y especialistas, la observación y las pruebas. Además para el proceso de desarrollo se utilizó RUP como metodología de software.

Como resultado final se obtiene un editor que permite diseñar interfaces gráficas de usuario, y un módulo que permite cargar, en proyectos que utilicen Ogre3D, el diseño previamente realizado. Esto brinda la posibilidad de desarrollar aplicaciones que dependan solamente de Ogre3D y Qt con los resultados visuales deseados.

Palabras Clave: diseñadores gui, interfaces gráficas de usuario, ogre3d, *overlay*.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Con el avance de las Tecnologías de la Informática y las Comunicaciones, casi todos los aspectos de la vida cotidiana han sido reflejados por la realidad virtual. Este fenómeno que ya cuenta con varios años de surgido ha dejado de ser una expresión ajena a lo común y tiene vastas aplicaciones tanto en los más avanzados centros científicos, como en los niveles más básicos de la enseñanza, tal es el caso de una propuesta novedosa en los medios de apoyo al aprendizaje conocida como laboratorios virtuales, cuyo objetivo no es más que facilitar la comprensión de los conceptos científicos a través de la simulación de estos en un computador.

El Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas se encuentra desarrollando varios laboratorios virtuales enmarcados en diferentes ramas de la ciencia. Para esta tarea, entre otras herramientas, utilizan el motor de *render Object-Oriented Graphics Rendering Engine* (OGRE), el cual es de un uso extendido por las prestaciones y documentación que brinda.

Esta herramienta es utilizada en el proceso de *render* de escenas 3D, por lo que, hasta versiones recientes, contaba con escasas opciones para el uso de menús. Esta deficiencia la solucionaba permitiendo el trabajo en conjunto con otros sistemas de interfaces gráficas de usuarios (GUI, acrónimo en inglés para *Graphic User Interface*). Entre los más conocidos de estos se encuentra CEGUI (siglas para *Crazy Eddie's* GUI), que normalmente era el que incluían consigo los ejemplos de Ogre3D en cada versión, aunque el uso de una mayor cantidad de estos sistemas de terceras partes, puede provocar retrasos en el desarrollo, porque requiere adquirir dominio de estas herramientas.

Entre las novedades de la versión 1.7.0 de Ogre3D se encuentra la desaparición de CEGUI de estos ejemplos, lo cual no quiere decir que no se pueda seguir utilizando este u otro sistema de interfaz gráfica; en su lugar, los ejemplos utilizan una clase que posibilita realizar menús gráficos muy simples, con componentes predeterminados construidos completamente con *overlays* de Ogre3D, por lo que no es necesario nada más que el propio motor.

Estos *overlays* son elementos bidimensionales que se posicionan encima del resto de la escena. La construcción de pequeños menús en aplicaciones, a partir de los componentes del nuevo sistema, es bastante sencilla; pero cuando estos tienden a ser más complejos, y se requieren en diversos escenarios, se pueden desencadenar errores o no obtener los resultados visuales deseados. Además, cada pequeño cambio que se necesite realizar en la interfaz implica recompilar toda la aplicación.

De aquí surge la siguiente interrogante, que es el problema de investigación a resolver, ¿Cómo diseñar interfaces gráficas de usuario utilizando las nuevas características de Ogre3D?

Como objeto de estudio se trabaja con los editores de interfaces gráficas de usuarios de ahí que el campo de acción sea los editores de interfaces gráficas de usuarios basadas en *overlays*. El objetivo principal que se propone en este trabajo es desarrollar un editor para una GUI constituida por los *overlays* de Ogre3D en conjunto con un módulo que permita cargar dicha GUI.

A continuación se plantean un grupo de tareas que permitirán satisfacer el objetivo:

- Elaborar el marco teórico para formalizar la investigación.
- Realizar un estudio del problema y de la situación actual del tema para determinar las características que debe tener el sistema.
- Analizar la nueva arquitectura de los laboratorios virtuales, para incluir el módulo de carga de la GUI de manera efectiva.
- Diseñar la solución de manera que se especifique como funcionará el sistema.
- Implementar el editor de GUI.
- Implementar el módulo de carga de la GUI.
- Validar el sistema realizado para determinar si cumple con el objetivo.

El presente trabajo está estructurado de la siguiente manera: Resumen, Introducción, cuatro capítulos de contenido, Conclusiones, Recomendaciones, Bibliografía Consultada y Glosario de Términos. A continuación se hace una breve descripción del contenido de los capítulos.

- Capítulo 1. Fundamentación Teórica.
- Capítulo 2. Características del Sistema. Descripción de las metodologías y herramientas empleadas. Descripción de la solución propuesta. Modelo del dominio; requisitos funcionales y no funcionales; actores y casos de uso del sistema.
- Capítulo 3. Diseño de la Solución. Descripción del diseño a través de diagramas de clases, descripción del flujo de eventos a través de diagramas de interacción.
- Capítulo 4. Implementación y Pruebas. Descripción de los elementos físicos del sistema a través de los diagramas de componentes. Validación de las funcionalidades del sistema.

Entre los métodos científicos a utilizar se destacan:

- Métodos Teóricos:
    - Analítico-Sintético: mediante su uso se van a analizar las informaciones obtenidas, para posteriormente realizar una síntesis de las mismas y arribar a las principales ideas.
    - Histórico-Lógico: mediante su uso se analizará la trayectoria y la evolución de los diferentes editores GUI existentes y el uso de los *overlays* en los mismos para determinar el más adecuado modo de empleo.
    - Modelación: se utilizará para crear un prototipo funcional del sistema.
- Métodos Empíricos:
    - Consulta de las fuentes de información: mediante el mismo se seleccionará la información necesaria para construir el marco teórico.
    - Consulta de especialistas: para recibir los criterios de validación sobre la aplicabilidad y utilidad del trabajo realizado.
    - Observación: para reunir información visual sobre lo que el objeto de estudio hace y cómo se comporta.
    - Pruebas: para valorar el desempeño del sistema elaborado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

El presente capítulo pretende mostrar los precedentes de este trabajo, comenzando desde los conceptos más generales y globales hasta terminar en aquellos términos que directamente tienen relación con el tema, de esta manera se sentarán las bases y se facilitará el entendimiento de los capítulos posteriores.

1.1 Acercamiento a las interfaces.

En el campo de la informática es usual escuchar el término interfaz sin importar el tema del que se esté tratando, esto se debe a que la palabra interfaz, por decirlo de algún modo, engloba un significado bastante amplio, tanto es así que puede definirse como “conexión o frontera común entre dos aparatos o sistemas independientes. (1)”

Este concepto se ha definido ampliamente desde diferentes puntos de vista, etimológicamente hablando la palabra interfaz tiene su base en las raíces latinas *inter* que significa “entre” o “en medio”, y *facíes* que significa “superficie” o “vista o lado de una cosa”, por lo que se podría definir como “superficie vista, o lado mediador”.

1.2 La interfaz de usuario.

La semántica de esta composición sugiere adicionar a la definición de interfaz, que una de las dos partes comunicantes sea necesariamente usuario de la otra. Por tanto la interfaz de usuario es el “Conjunto de elementos a través de los cuales un usuario interactúa con un objeto que realiza una determinada tarea. (2)”

En informática una “interfaz de software es la parte de una aplicación que el usuario ve y con la cual interactúa. Está relacionada con la subyacente estructura, la arquitectura, y el código que hace el trabajo del software, pero no se confunde con ellos. La interfaz incluye las pantallas, ventanas, controles, menús, metáforas, la ayuda en línea, la documentación y el entrenamiento. Cualquier cosa que el usuario ve y con lo cual interactúa es parte de la interfaz. Una interfaz inteligente es fácil de aprender y usar. Permite a los usuarios hacer su trabajo o desempeñar una tarea en la manera que tiene más sentido para ellos, en vez

de tener que ajustarse al software. Una interfaz inteligente se diseña específicamente para la gente que la usará. (3)"

Las interfaces de usuario han estado ligadas desde su surgimiento a los sistemas operativos imperantes, y han sido la causa principal del éxito de algunos frente a otros, en un primer lugar en esta cadena evolutiva se encuentran las interfaces de líneas de mandatos (*Command-Line User Interface*, CUIs), este tipo de interfaz constituye un lenguaje formal con vocabulario y sintaxis propia, su hardware de interacción no es más que el teclado y responde al principio de orden-acción. (2)

En un segundo lugar se encuentran las interfaces de menús que no son más que una lista de opciones que se muestran en pantalla o en una ventana de la pantalla para que los usuarios elijan la opción que deseen y permiten navegar dentro de un sistema y seleccionar elementos de una lista, que representan propiedades o acciones que los usuarios desean realizar. (2)

Como evolución lógica de las interfaces de menús surgen las interfaces gráficas de usuario, desarrolladas originalmente por Xerox (Xerox Star, 1981) y Popularizadas por Apple (Apple Lisa, 1983) (Apple Macintosh, 1984); cuando Star fue introducida en 1981, su monitor de mapas de bits, las ventanas, el mouse y los iconos eran únicos en el mercado. (4)

1.3 La interfaz gráfica de usuario.

Una GUI no es más que una representación gráfica en pantalla de los programas, datos y objetos que posee entre sus características, aplicar el paradigma de interacción objeto-acción, posee un dispositivo apuntador y además promueve la consistencia de la interfaz entre programas. (2)

Por otra parte las interfaces gráficas pueden definirse, "en el contexto de la interacción persona-ordenador, como un artefacto interactivo, que por su diseño y a través de ciertos interfaces humanos, posibilita la interacción de una persona con el sistema informático, haciendo uso de las gramáticas visuales y verbales. (5)"

En general una GUI es un tipo específico de interfaz, que utiliza metáforas visuales y gráficas como paradigma interactivo entre el usuario y el ordenador. (5)

Existen tres estilos principales de interfaces gráficas hombre-ordenador, estas son: (3)

1. Lo que tú vez es lo que puedes conseguir (WYSIWYG, *What you see is what you get*).
2. Manipulación directa.
3. Interfaces de usuario basadas en íconos.

Técnicamente, todos los elementos en la pantalla, en las interfaces más comunes, son ventanas con características que las hacen apropiadas para el uso que se le esté dando, algunas de estas son grandes, otras pequeñas y algunas contienen otras ventanas. Por ejemplo, un botón podría ser una pequeña ventana que tiene un borde y un texto en el mismo, una barra de desplazamiento podría ser una ventana que tiene pequeños botones en forma de triángulo y un control deslizante, una ventana desplegable además de sus barras de desplazamiento es en sí un elemento que sabe cómo lidiar con los cambios de contenido y el control deslizante. Los botones y el cierre son también ventanas. Cuando se hace clic en un botón, este es reemplazado por otra ventana temporal que simula ser un botón pulsado. (7)

1.4 Editores de GUI.

Un editor de interfaces gráficas de usuario, también conocido como constructores de GUI (GUI *builder*) o diseñadores de GUI (GUI *designer*), es una herramienta de software que simplifica la creación de GUI permitiéndole al diseñador usar un editor WYSIWYG para arrastrar y soltar componentes hasta obtener lo deseado. Sin un Editor GUI estas deben ser construidas manualmente especificando los parámetros de cada elemento mediante código, sin ninguna retroalimentación hasta que la aplicación no esté corriendo.

En general los editores de GUI, proveen una abstracción de las librerías de diseño gráfico, para desarrollar en forma más sencilla las interfaces. Los diseñadores de GUI modernos utilizan, a su vez, una interfaz gráfica para comunicarse con el desarrollador. Es decir, el desarrollador crea su interfaz mediante el lenguaje gráfico proveído por la herramienta (paletas de componentes, acciones de “arrastrar” y “soltar”, presión de botones, selectores de colores, etc.). No obstante, dichas herramientas también requieren el uso de texto para ajustar en forma fina el comportamiento y las propiedades de los componentes.(6)

1.4.1 Partes usuales de los editores de GUI.

En general lo que se denomina ventana de la aplicación tiene una barra de título en la parte superior, con iconos para cerrar y cambiar su tamaño. En la propia ventana se puede ver una barra de menús con

desplegables, ventanas con alguna información, las barras de desplazamiento, botones y una barra de estado en la parte inferior. (7)

Crear una aplicación mediante la agrupación de ventanas como se mencionaba anteriormente sería un trabajo tedioso, para ello, los editores de GUI proporcionan elementos pre-construidos llamados *widgets*, que convierten el proceso de construir una aplicación con interfaz gráfica en un trabajo mucho más fácil. (7)

Por esta razón los editores de GUI poseen una paleta con los diversos componentes clasificados según las distintas áreas a las que esté orientado el mismo, una de las facilidades que poseen los editores actuales más potentes es la posibilidad de programar componentes propios con características específicas deseadas por el diseñador, lo cual brinda una gran flexibilidad. (6)

La ventana de propiedades es otra de las secciones que poseen por lo general los editores GUI, la misma permite cambiar el diseño estándar de un componente para personalizarlo. Todos los cambios que se realizan en la interfaz, a la vez, son mostrados en este menú. (6)

Por lo general también se brindan características como la previsualización de la interfaz. En la actualidad es usual el uso del formato Lenguaje de Marcas Extensibles (*Extensible Markup Language*, XML) para guardar la descripción de la interfaz. (6)

En el ámbito de este trabajo se denomina a estas descripciones “configuración”.

1.4.2 Ventajas de los editores de GUI.

Los editores de interfaces gráficas de usuario en general tienen las siguientes ventajas: (6)

- Permiten desarrollar rápidamente una GUI, especialmente si se considera el tiempo que se requeriría para realizar dicha interfaz mediante instrucciones del lenguaje de programación.
- Si la herramienta de diseñado de GUI es de uso muy extendido, las GUI producidas por ésta seguirán un formato gráfico semejante, lo que hará que para muchos de los usuarios finales sea intuitivo utilizar la nueva aplicación.
- El lenguaje gráfico utilizado para la generación de interfaces normalmente sigue muy de cerca o es igual al lenguaje gráfico que ofrecerán las GUI por dicha herramienta diseñadas. Esto permite que,

al desarrollar las interfaces gráficas, el desarrollador se familiarice a su vez con el uso de los distintos componentes que puede utilizar en su aplicación y pueda evaluar el efecto que los mismos producirán en sus usuarios finales.

- El uso extendido de los editores de GUI ha hecho que el lenguaje gráfico utilizado por estos sea muy semejante, esto quiere decir que los diversos generadores de GUI existentes por lo general ofrecen componentes gráficos equivalentes con nombres semejantes (menús, botones, etiquetas, listas seleccionables y otros). Esto ha producido que cada vez más haya un grupo importante de usuarios para los que dicho lenguaje es conocido.

1.4.3 Desventajas de los diseñadores de GUI.

Como se ha indicado, los diseñadores de GUI tienen grandes ventajas. No obstante, también su uso presenta importantes inconvenientes: (6)

- Los diseñadores de GUI son altamente pre-configurados. Como lo que se busca es indicar el qué hay que hacer y no el cómo, el diseñador de pantallas implícitamente reduce el dominio de lo que el lenguaje de programación y las librerías subyacentes pueden lograr. Por ejemplo, que los botones fueran redondos en vez de rectangulares, utilizar componentes con animaciones 3D, etc.
- Si bien los diseñadores pueden permitir establecer código antes y después de la creación de los componentes, esto no es diferente a realizar la tarea directamente en el lenguaje de programación e incluso puede ser más engorroso porque hay que determinar como “deshacer” u obviar el código que el generador crea por defecto.
- Todos los generadores están asociados a uno o más lenguajes de programación predeterminados e inclusive pueden estar asociados a una infraestructura o plataforma determinada. Esto hace que estas poderosas herramientas no puedan ser aplicadas en forma general, para resolver problemas con lenguajes de programación no soportados.
- Estas herramientas reducen la interfaz con el usuario a una interfaz gráfica, no obstante existen muchos medios de percepción, por ejemplo el sonido, que podrían utilizarse para brindar accesibilidad a personas con alguna necesidad especial (daltónicos, problemas visuales, etc.) Y, en todo caso, si cualquier persona percibe una información por diversos sentidos, se da un

procesamiento más eficiente de la misma a nivel cerebral. El uso de una herramienta de creación de GUI brinda poco soporte para la creación de otras interfaces (braille, sonido, etc.) y la mezcla de interfaces producidas por diversas herramientas provoca un problema complejo de programación y sincronización.

- Otro aspecto importante es que, si bien los creadores de interfaces gráficas ofrecen algún grado de apoyo a la internacionalización de sus aplicaciones, éste casi que se refiere a brindar la traducción de los elementos textuales. El problema con ello es que una internacionalización verdadera de una interfaz gráfica debe tomar en cuenta también la necesidad de cambiar de posición los elementos de la interfaz. (8)
- En general cuando se utiliza un diseñador de GUI, el desarrollador de software se está atando al paradigma de lenguaje gráfico que dicho generador provee. Y si dicho desarrollador requiere algo que extienda o esté fuera de dicho paradigma va a tener que realizar un esfuerzo importante para poder lograrlo.

1.4.4 Una mirada a varios diseñadores GUI.

Microsoft Windows y Mac OS tienen sus propios conjuntos de herramientas (*toolkits*) que pueden utilizarse para las aplicaciones GUI. Para el X Window System existen muchos *toolkits* de *widgets* populares, como Qt (KDE) y GTK+ (GNOME). Otro conjunto de herramientas importante es el *toolkit* wxWidgets que proporciona un conjunto multiplataforma de *widgets* implementados en X, Motif, GTK, Mac OS, Palm OS, Microsoft Windows, y varios más. (7)

Varios ejemplos de diseñadores son los siguientes:

API Cocoa:

Cocoa es la interfaz de programación de aplicaciones (API, por sus siglas en inglés) nativa de Apple orientado a objetos para el sistema operativo Mac OS X. La utilización con la API Cocoa Touch incluye reconocimiento gestual, animación, y una biblioteca de interfaz de usuario diferente, y es para aplicaciones del sistema operativo IOS, que se utiliza en los dispositivos de Apple como el iPhone, el iPod Touch y el iPad. (9)

Usando la API Cocoa se encuentra la herramienta:

1. Interface Builder

Interface Builder ofrece paletas, o colecciones, de objetos de la interfaz de usuario para un desarrollador de Objective-C. Estos objetos de interfaz de usuario contienen elementos como campos de texto, tablas de datos, deslizadores y menús pop-up. Las paletas de Interface Builder son totalmente extensibles, lo que significa que cualquier desarrollador puede desarrollar nuevos objetos y agregar paletas a Interface Builder.

Para crear una interfaz, un desarrollador, simplemente arrastra objetos de interfaz de la paleta a una ventana o un menú. Las acciones que los objetos pueden emitir están conectadas a los objetivos en el código de la aplicación. (9)

GTK+:

GTK+ (GIMP *Toolkit*) es un *widget toolkit* multiplataforma para crear interfaces gráficas de usuario. Es una de las herramientas más populares para el X Window System. Está orientado a objetos, escrito en el lenguaje de programación C. Aunque GTK + está dirigida principalmente a X Window System funciona en otras plataformas, incluyendo Microsoft Windows (a través del API de Windows), y Mac OS X (a través de Quartz). (10)

Usando GTK+ se encuentra la herramienta:

2. Glade Interface Builder

Glade Interface Builder posee componentes adicionales para GNOME. En su tercera versión, Glade es independiente del lenguaje de programación, y no produce código para los eventos, sino más bien un archivo XML que se utiliza con un enlace apropiado (como por ejemplo GtkAda para su uso con el lenguaje de programación Ada). (11)

Las interfaces de usuario diseñadas en Glade se guardan como archivos XML, y mediante el uso de GtkBuilder estos pueden ser cargados por aplicaciones de forma dinámica según sea necesario. Mediante el uso de GtkBuilder, los archivos XML de Glade se puede utilizar en numerosos lenguajes de programación como C, C++, C #, Vala, Java, Perl, Python y otros. (11)

Qt:

Qt es un *framework* de aplicaciones multiplataforma que se utiliza ampliamente para el desarrollo de aplicaciones de software con una interfaz gráfica de usuario (en este caso Qt es clasificado como un *widget toolkit*), y también se utiliza para el desarrollo de programas sin interfaz gráfica.

Qt es desarrollado por un proyecto de código abierto, el Qt Project, que implica tanto a los desarrolladores individuales así como desarrolladores de Nokia, Digia, y otras compañías interesadas en su desarrollo. (12)

Qt utiliza el estándar C++, pero hace un amplio uso de un generador de código especial (llamado el compilador Meta Object, o MOC), junto con varias macros para enriquecer el lenguaje. Qt también se puede utilizar en varios lenguajes de programación. Se ejecuta en las plataformas de escritorio más importantes y algunas de las plataformas móviles.

Distribuido bajo los términos de la GNU *Lesser General Public License*, entre otros, Qt es un software libre y de código abierto. Todas las ediciones soportan un amplio rango de compiladores, incluido el GCC, C++ y la suite de Visual Studio.

Usando Qt se encuentra la herramienta:

3. **Qt Designer**

Qt Designer permite crear interfaces de usuario con herramientas de diseño que mueven y escalan los *widgets* de forma automática en tiempo de ejecución. Las interfaces resultantes son a la vez funcionales y atractivas, satisfaciendo los requerimientos operativos de los usuarios y las preferencias. Qt Designer incluye un editor de código que se puede utilizar para integrar los espacios personalizados en el interior del código generado. (13)

wxWidgets:

El wxWidgets es otro *widget toolkit* y una biblioteca de herramientas para crear interfaces gráficas de usuario para aplicaciones multiplataforma. El wxWidgets le permite al código de un programa ser compilado y ejecutado en varias plataformas de ordenador con cambios en el código mínimo. Cubre los sistemas como Microsoft Windows, Mac OS X (Carbon y Cocoa), iOS (Cocoa Touch), Linux / Unix (Motif X11, y GTK +), OpenVMS, OS / 2 y AmigaOS. (14)

Usando wxWidget se encuentra la herramienta:

4. **wxGlade**

wxGlade puede generar código de diseño para C++, Lisp, Python y Perl. Es un diseñador de interfaz gráfica de usuario escrito en Python. El código generado por esta herramienta no realiza ninguna acción, aparte de mostrar los *widgets* creados. (7)

FLTK:

FLTK es un *toolkit* GUI multiplataforma en C++ para UNIX / Linux (X11), Microsoft Windows y MacOS X que proporciona funcionalidad GUI sin complicaciones y es compatible con gráficos 3D a través de OpenGL y la GLUT. (15)

El mismo está diseñado para ser pequeño y modular, lo suficiente como para ser enlazado estáticamente, pero funciona bien como una biblioteca compartida, incluye un editor de interfaz de usuario excelente llamado "FLUID" que se puede utilizar para crear aplicaciones en minutos. (15)

CEGUI:

Crazy Eddie's GUI System, es una biblioteca escrita en C++ para interfaces gráficas de usuario. Está diseñado especialmente para videojuegos, pero se puede utilizar para otras tareas. (16)

La fuerza del diseño de CEGUI es que es altamente configurable. El sistema CEGUI en sí mismo no carga directamente los archivos o ventanas de *render*. CEGUI interactúa con éstos a través de código definido por el usuario, aunque CEGUI viene con una serie de módulos para el uso de ciertos componentes y bibliotecas. (16)

Esta libertad permite al usuario utilizar CEGUI en cualquier tipo de sistema de gestión de los recursos o ambiente operativo. Las entradas se espera que sean recogidas por el código del usuario, posiblemente filtrada como el usuario considere conveniente, y luego entregado a la CEGUI para el procesamiento de ventana, además posee un conjunto razonable de *widgets*. (16)

Usando CEGUI se encuentra la herramienta:

5. **CELayoutEditor**

Permite crear una interfaz con métodos similares a los entornos de desarrollo visual. Como características se pueden listar las siguientes: (16)

- Las ventanas y objetos pueden ser colocados en una jerarquía.
- Modo de posicionamiento absoluto y relativo.
- Utiliza el paradigma WYSIWYG.
- Guarda el diseño en un archivo XML, listo para usar en CEGUI.

1.5 Un acercamiento a las *widgets* más comunes.

Múltiples son los *widgets* existentes debido a la gran cantidad de bibliotecas gráficas que existen, pero con pequeñas variaciones prácticamente todas estas bibliotecas contienen los siguientes.

Botones de comando (*buttons*): Los botones constituyen dentro de los cuadros de diálogos la principal manera que tiene el usuario para realizar acciones. Algunas etiquetas se han convertido en estándares para las aplicaciones como por ejemplo: Aceptar, Cancelar, Ayuda.

Botones de opciones (radio *buttons*): Los botones de opciones son utilizados para cuando se vaya a escoger una opción entre varias que brinda la aplicación.

Casillas de verificación (*check box*): Si lo que se desea es escoger una o varias opciones de sí o no, en lugar de usar *radio buttons* es mejor utilizar *check boxs*, los cuales se activan o desactivan.

Cajas de texto (*edit box*): Las cajas de texto constituyen la principal vía mediante la cual el usuario ingresa información a través del teclado a una aplicación. Los cuadros de texto pueden tener asignados una etiqueta que muestre el contenido que el usuario debe teclear.

Etiqueta (*label*): Son la vía mediante la cual se suele indicar la función de otra de las *widgets*, contienen un texto estático para esto.

1.6 Sistema de *overlays* de Ogre3D.

Los *overlays* son objetos de Ogre3D que permiten generar elementos en 2 y 3 dimensiones encima de los contenidos de la escena para crear efectos como sistemas de menú o paneles de estado. (17)

Estos pueden ser creados a través del Manejador de Escena (SceneManager) de Ogre3D, o se pueden definir en un *script* propio con extensión .*overlay*, en la práctica el segundo método es probable que sea el

más usado porque es más fácil de ajustar, sin la necesidad de recompilar el código. Se debe tener en cuenta que se pueden definir tantas capas como se quiera; todos los *overlays* comienzan estando ocultos, y son mostrados a medida que se necesite. La superposición de los mismos está determinada por la propiedad llamada zOrder. (17)

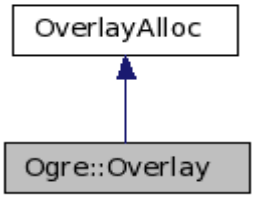

Figura 1: La clase *overlay*

La clase OverlayElement de Ogre3D, abstrae los detalles de elementos 2D que se añaden a los *overlays*. Todos los elementos que se pueden añadir a las plantillas se derivan de esta clase. Es posible para los usuarios de Ogre3D definir sus propias subclases personalizadas de OverlayElement con el fin de proporcionar sus propios controles de usuario. Las principales características comunes de todos los OverlayElement son cosas como el tamaño, la posición y el nombre del material básico. Las subclases extienden este comportamiento para incluir las propiedades más complejas y definir su propia rutina. (17)

Entre las subclases de los OverlayElement se encuentra el OverlayContainer, este puede contener otros OverlayElement, proporcionándoles un origen de coordenadas local para facilitar la alineación y a su vez los agrupa, lo que permite que se puedan tratar como un conjunto. De hecho no todas las instancias de OverlayElement se pueden agregar directamente a un *overlay*, sólo aquellos que son instancias de este descendiente.

Otra de las subclases de los OverlayElement es el TextAreaOverlayElement, este está diseñado para mostrar texto encima del *overlay*.

La cuarta clase de importancia es OverlayManager. Cuando una aplicación desea crear un elemento 2D para añadirlo a un contenedor, se debe utilizar dicha entidad. (17)

Los *overlays* sólo son realmente diseñados para elementos de pantalla no interactivos, aunque se puede utilizar como una GUI muy básica. Para una solución de GUI más completa, es usual utilizar CEGUI.

Las relaciones entre estas clases se pueden apreciar a continuación.

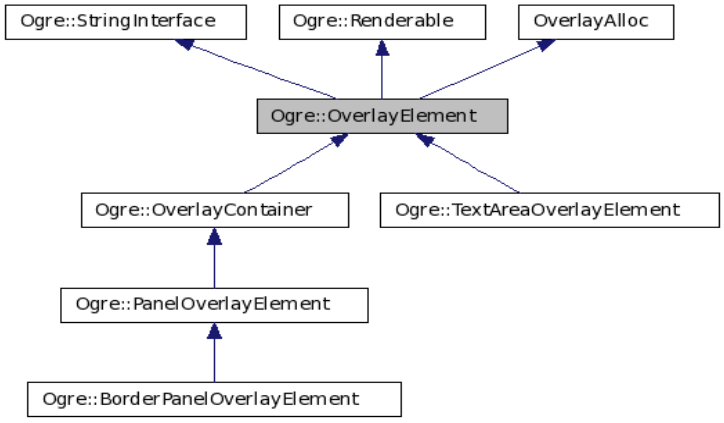

Figura 2: Relaciones entre las clases de *overlay*

Utilizando los OverlayElement y el resto de sus clases hijas es posible realizar elementos tan complejos como sea necesario, anidando unos dentro de otros, como muestra el ejemplo de la siguiente imagen.

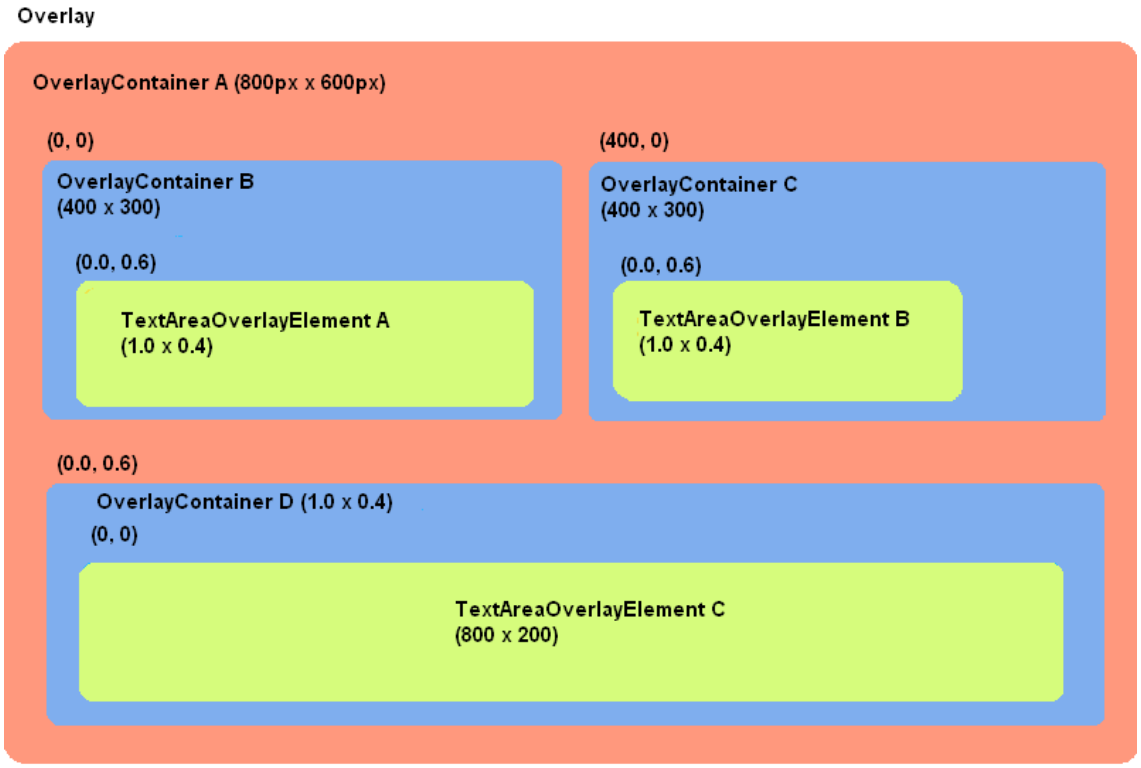

Figura 3: Estructura jerárquica de un *overlay*

1.7 SdkTrays.

El sistema SdkTrays fue creado para tener controles GUI muy simples e independizar las interfaces gráficas que poseen los ejemplos que brinda Ogre3D, de CEGUI; el mismo está basado en el sistema de *overlays* de dicho motor de *render*. Su simpleza le impone varias limitaciones pero es lo que lo convierte en un sistema muy fácil de usar.

El concepto de bandejas (*trays*) que provee es fundamental para el sistema ya que elimina la necesidad por parte de un programador de calcular y especificar las coordenadas de posición de cualquier tipo. Por lo general las ubicaciones que se le dan a las *widgets* en las GUI típicas, son cualquiera de las cuatro esquinas, a lo largo de cualquiera de los cuatro bordes, o en el centro de la pantalla. Por esta razón SdkTrays provee una "bandeja" para cada uno de los lugares anteriormente mencionados. Cuando se crea un *widget*, se especifica una de estas nueve localidades, y este se agrega a la bandeja correspondiente. El *widget* se añadirá directamente debajo del último que fue añadido a la misma bandeja. Al realizar esta operación la bandeja reajusta su propio tamaño. (18)

SdkTrayManager es la clase a través de la cual se crean y gestionan todos los *widgets*, también permite manipular el cursor, cambiar la imagen de fondo, ajustar las propiedades de las bandejas, los cuadros de diálogo, mostrar u ocultar la barra de carga, y más. Para utilizar SdkTrays debe primero cargarse el recurso "SdkTrays.zip". (18)

A continuación se muestra una breve descripción de los principales componentes de SdkTrays:

1 Cursor

El cursor consta de tres partes, en primer lugar está la capa en que se mueve que constituye un *overlay*, el contenedor en que se coloca el cursor, o sea, un OverlayContainer, y la imagen del cursor, que es un elemento secundario del contenedor, específicamente un OverlayElement. Al ocultar o mostrar el cursor, lo que se manipula es el *overlay* que lo contiene. (18)

2 Telón de fondo (*Backdrop*)

Si se desea mostrar un telón de fondo, existe una funcionalidad que toma el nombre de un material opcional para este. Si no se especifica, la imagen de fondo no se cambia. (18)

3 *Widgets*

El sistema provee 10 *widgets* básicas, cada una de estas es solo una instancia de una plantilla de OverlayElement y usan métrica de pixel por defecto. Como los *overlay*, requieren un nombre único dentro de Ogre3D, al crearlas es necesario especificar la bandeja en que serán colocadas. (18)

3.1 Botón (*Button*)

Este es el *widget* más básico de todos. Debe especificarse un título y un ancho opcional. Si el ancho no se especifica, este cambia automáticamente el tamaño para adaptarse a su título. (18)

3.2 Caja de texto (TextBox)

Este control consiste en una barra de título en negrita y un área de texto que se puede desplazar verticalmente. Se le debe especificar un título, ancho y alto. Se puede tener acceso al título, el contenido del área de texto, el relleno de texto, la alineación del texto, o cambiar manualmente la posición de la barra de desplazamiento, entre otras cosas. (18)

3.3 Menú de selección (SelectMenu)

Es un menú desplegable básico en el cual se puede acceder al título, y a los elementos de muchas maneras diferentes. También puede cambiar manualmente la selección, con la opción de no disparar ningún evento. (18)

3.4 Etiqueta (*Label*)

Este *widget* utiliza una fuente diferente del resto, y es bueno para los encabezados de sección. Requiere un título, y una anchura opcional. Si el ancho no se especifica la etiqueta ajusta automáticamente su tamaño para llenar la bandeja en que sea ubicada. (18)

3.5 Separador (*Separator*)

No es más que una línea horizontal útil para dividir otros *widgets* en secciones sin ocupar demasiado espacio. Necesita una anchura opcional, si no se especifica el comportamiento es el mismo que el de las etiquetas. (18)

3.6 Control deslizante (*Slider*)

Una barra de desplazamiento consiste en una caja contenedora, un pequeño cuadro de texto para mostrar el valor, y un manejador de la pista. Requiere tres parámetros: un valor mínimo, un valor máximo, y el número de "puntos intermedios". (18)

3.7 Panel de parámetros (ParamsPanel)

Este panel muestra un número arbitrario de parámetros y sus valores, los nombres de los parámetros serán justificados a la izquierda, y sus valores se justifican a la derecha en la misma línea. (18)

3.8 Caja de chequeo (CheckBox)

Necesita un título y ancho opcional, no especificar el ancho significa ajuste automático al título. Se tiene acceso al estado de la casilla de verificación, con la opción de no desencadenar un evento. (18)

3.9 Ventana decorativa (DecorWidget)

Este *widget* toma cualquier plantilla de OverlayElement y crea un *widget* de ella. Si se quiere poner una imagen, iconos, u otros objetos estáticos, se debe hacer una plantilla de OverlayElement en un *script*, y luego convertirlo en un DecorWidget. (18)

3.10 Barra de progreso (ProgressBar)

Posee un título, cuadro de detalles, y un medidor que se llena. Además del título requiere un ancho total, y el ancho del cuadro de comentario. (18)

4 La bandeja nula (*Null Tray*)

Además de las nueve bandejas en que se pueden colocar los *widgets*, este sistema también provee una "bandeja nula", la cual no es visible como tal, su localización se especifica mediante el enumerador de posición TL_NONE y no dispone los *widgets* en su interior. La misma está creada para colocar de forma manual un *widget* en cualquier lugar de la pantalla. (18)

5 SdkTrayListener

Esta clase contiene los controladores para todos los diferentes eventos que pueden disparar los *widgets*. La clase SdkTrayManager es en sí un SdkTrayListener, porque contiene varios *widgets* especiales que también lanzan eventos. (18)

Para responder a los eventos que disparan los *widgets* es necesario heredar de esta clase, de manera que se tendría acceso al componente que disparó el evento. Los eventos que esta incluye son los siguientes: (18)

- buttonHit: Devuelve un puntero al botón que se vio afectado.
- itemSelected: Devuelve un puntero al tema del SelectMenu que fue elegido. A continuación, puede utilizar el SelectMenu para ver qué opción ha sido seleccionada.
- labelHit: Devuelve un puntero al label que se vio afectado.
- sliderMoved: Devuelve un puntero al deslizador cuyo valor se ha cambiado. A continuación, puede utilizar el control deslizante para ver cuál es su nuevo valor. Además, puede convertir este valor en una forma más adecuada y mostrarlo.
- checkBoxToggled: Devuelve un puntero a la casilla de verificación cuyo estado ha cambiado. A continuación, se puede utilizar la casilla para ver si ha sido activada o desactivada.
- okDialogClosed: Devuelve el mensaje del diálogo que fue cerrado.
- yesNoDialogClosed: Devuelve la pregunta del diálogo que fue cerrado, y un valor booleano indicando si la opción escogida fue “Si” o “No”.

1.8 GuiSystem.

GuiSystem es una biblioteca desarrollada por el proyecto de los laboratorios virtuales a partir del SdkTrays, este es el sistema para el cual se desarrollará un editor de configuraciones y un módulo que permita cargar las mismas; constituye una encapsulación de las distintas clases de SdkTrays y además agrega varias funcionalidades y nuevas *widgets*, además de disparar nuevos eventos. A continuación se enumeran sus nuevas características.

1 Caja de edición (EditBox)

Este *widget* posee una etiqueta que identifica cual será su uso y además una caja de texto editable que puede ser entrado el presionar la tecla “Enter”. Para su creación necesita de un título y ancho.

2 Bandeja (*Tray*)

Las bandejas permiten anidar tantas *widgets* como se desee en su interior, incluso más bandejas; utilizando este elemento se puede agrupar los *widgets* en pequeños grupos para tener un diseño más organizado. Requiere solamente de sus dimensiones y no puede ser colocado dentro de las bandejas predefinidas.

3 **editboxSelected:** Devuelve un puntero a la caja de edición que se va a modificar.

4 **editboxDeactivated:** Devuelve un puntero a la caja de edición que se acaba de ser modificada.

Conclusiones generales del capítulo.

Las descripciones de las distintas herramientas existentes para editar interfaces gráficas de usuario, plasmadas en este capítulo, permiten derivar características que el software que se diseñará en los capítulos siguientes debe tener, por otra parte la descripción detallada de la clase SdkTrays, la biblioteca GuiSystem y los overlays de Ogre3D permite conocer la estructura del sistema sobre el que se pretende desarrollar la solución, por lo que es vital para el éxito de la misma.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción.

En este capítulo se hará referencia a la propuesta de solución para el problema que concierne al presente trabajo de diploma, en general se dará una breve descripción de la misma, además de las tareas que engloban la determinación de un modelo del negocio y la captura de requisitos del sistema.

2.1 Solución propuesta.

Para darle solución al problema planteado se propone desarrollar dos sistemas que trabajarían en conjunto para obtener los resultados deseados, el primero de estos sistemas es un editor de GUI, y el segundo una clase que permite cargar GUI. El flujo dentro del proceso de construir una interfaz gráfica de usuario a partir de estos sistemas consta de dos momentos fundamentales, primeramente un diseñador de interfaces gráficas de usuario utilizando el editor realizaría un diseño que exportaría tras quedar satisfecho con la apariencia del mismo, en un segundo momento un desarrollador de los Laboratorios Virtuales utilizando una instancia de la clase de cargar GUI, levantaría el diseño y reimplementaría los métodos de la clase widgetListener para enlazar los componentes, esta clase es el equivalente en el GuiSystem a SdkTrayListener en SdkTrays. Al reimplementar de estos métodos los que sean necesarios, de acuerdo a los componentes con que cuente el diseño, se tendría acceso al estado de estos, siempre que lance un evento.

Solo falta resaltar que el editor de configuraciones de GUI debe permitir diseñar la interfaz que el diseñador desee a partir de los componentes disponibles en la biblioteca, para luego exportar este diseño. Por la claridad y facilidades que brinda al usuario se escoge el paradigma WYSIWYG para la construcción de este editor, además aquellos recursos que sean necesarios para realizar una configuración determinada deben ser cargados dentro de un paquete con extensión “zip”, y una vez que vayan a ser cargadas las GUI en la aplicación, los recursos que esta necesite también deben haber sido cargados.

2.2 Funcionalidades.

El editor de configuraciones de interfaz gráfica de usuario para los laboratorios virtuales debe contar con un conjunto de requerimientos funcionales y no funcionales que lo convierten en el producto que debe ser.

Como parte de los requerimientos funcionales se han identificado los siguientes, gestionar una configuración, lo cual incluye adicionar nuevas *widgets*, modificarlas dentro de la configuración o eliminarla si es lo que se desea, estas modificaciones pueden ser de varios tipos entre ellas cambiarlas de posición, de tamaño o anidarlas una dentro de otra, en el caso de aquellos componentes que lo permitan, además debe posibilitar obtener la información de las propiedades de un *widget* determinado de la configuración así como modificar dichas propiedades de manera que este cambio se manifieste al momento. El sistema debe permitir por otra parte pre-visualizar como quedaría el sistema diseñado una vez que esté corriendo en la aplicación. Además es indispensable que permita exportar dicha configuración en un formato factible y también importar una configuración creada con anterioridad.
Por otra parte el módulo de carga debe permitir cargar estas configuraciones a través del GuiSystem.

2.3 Principios de diseño empleados y herramientas utilizadas.

En el capítulo anterior se hacía mención a los tipos de interfaces de usuario existentes, de estas se determinó que era conveniente utilizar las interfaces gráficas debido a las grandes ventajas que posee respecto a las restantes en el problema que se está tratando.
Se seleccionó para el diseño del editor, características encontradas en herramientas similares que cuentan con gran aceptación, para facilitar la familiaridad. Por esta razón la ubicación es la que sigue, a la izquierda de la pantalla, queda ubicado un menú que muestra las distintas *widgets* que posee la biblioteca, a la derecha, el menú de propiedades que se muestra en caso de seleccionar algún *widget* de forma individual, quedando el centro, como área de trabajo principal, aunque puede utilizarse para esto cualquier región, por último, encima se encuentra el menú principal el cual brinda acceso a varias de las funcionalidades. A continuación se muestra un prototipo no funcional de la propuesta de solución.

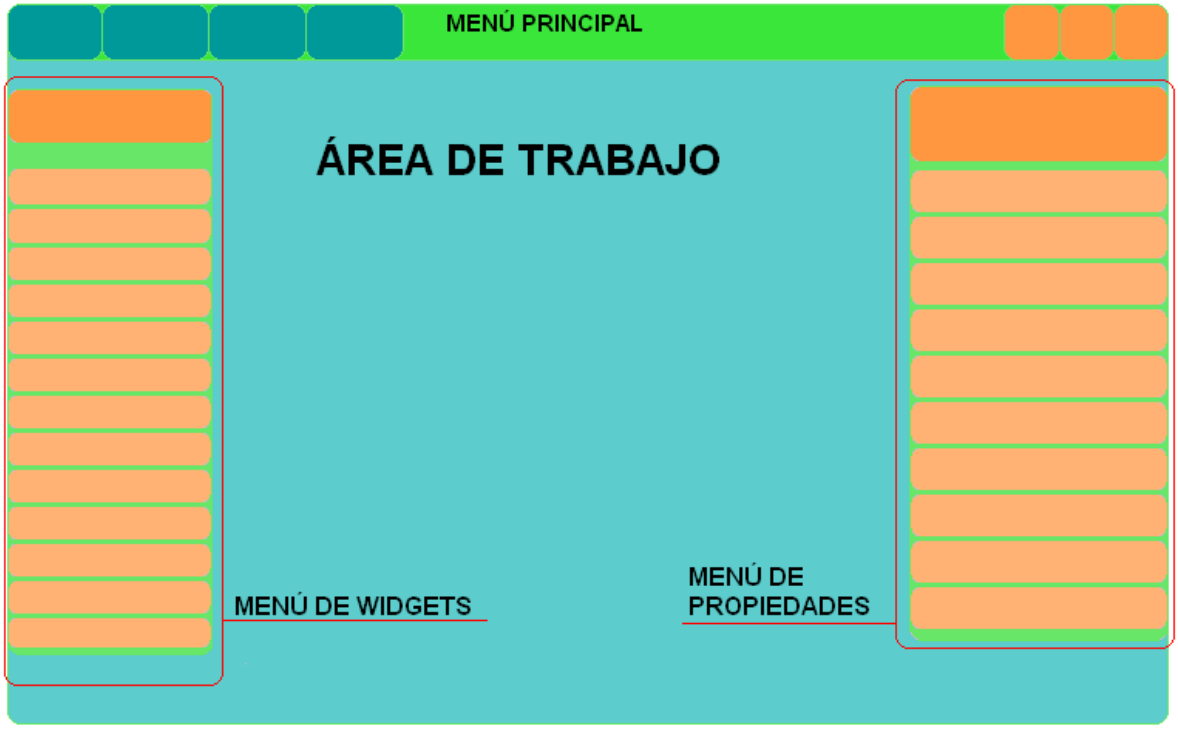

Figura 4: Prototipo no funcional de la propuesta de solución

2.3.1 Bibliotecas gráficas.

Luego de un estudio teórico realizado, se optó por desarrollar el menú principal utilizando el *framework* Qt, ya que este es totalmente orientado a objeto y su código está implementado directamente en C++, la calidad en tiempo de ejecución es muy alta, y no necesita de otras herramientas ni otra biblioteca. Posee un uso adecuado de la memoria y es multiplataforma. Presenta una muy buena documentación y ayuda online en internet, así como foros. (12)

Por otra parte se utilizará para los menús de *widgets* y propiedades la biblioteca de componentes GuiSystem, potenciando la familiaridad con el resultado final que obtendrá el usuario que emplee la herramienta.

2.3.2 Entorno de desarrollo Integrado

Después de haber estudiado varios Entornos de Desarrollo Integrado se escogió el Qt Creator para el desarrollo de los Laboratorios Virtuales ya que presenta características imprescindibles como un editor avanzado de código C++, posee herramientas para la administración de proyectos, un depurador visual,

un GUI integrado además de resaltado y autocompletado de código que lo convierte en una de las mejores herramientas para desarrollar con las bibliotecas de Qt.

2.3.3 Metodología

El Proceso Unificado de Racional (RUP por sus siglas en inglés), es una metodología que permite el desarrollo de software a gran escala, mediante un proceso continuo de desarrollo. En la misma se modelan visualmente los productos de software empleando el Lenguaje de Modelado Unificado (UML por sus siglas en inglés). RUP posee tres características fundamentales, es dirigido por casos de uso, lo cual implica que desde la captura de requisitos a través del usuario, los mismos sirvan de hilos conductores en todo el proceso de desarrollo. Otra característica fundamental es que está centrado en la arquitectura, la cual describe utilizando diferentes vistas; y finalmente, es iterativo e incremental por lo que divide el desarrollo en pequeñas partes consideradas cada una como iteraciones que al concluir aportan un incremento al producto final, las mismas son planificadas cuidadosamente desde el inicio del proceso.
Por la gran organización y potencia que ha demostrado esta será la metodología empleada en el presente trabajo.

2.4 Modelo de dominio.

El negocio se representa mediante un modelo de dominio, ya que es fácil de interpretar el funcionamiento del sistema y la simplicidad del entorno donde está enmarcado. Se escogió esta representación porque la misma es un subconjunto del modelo del negocio y porque en el caso que se enfrenta las fronteras del mismo no son claras. Como un mapa conceptual, este modelo, es una representación visual de las clases abstractas del mundo real. Su función es ayudar a comprender el problema que se plantea. Dicho modelo es el siguiente.

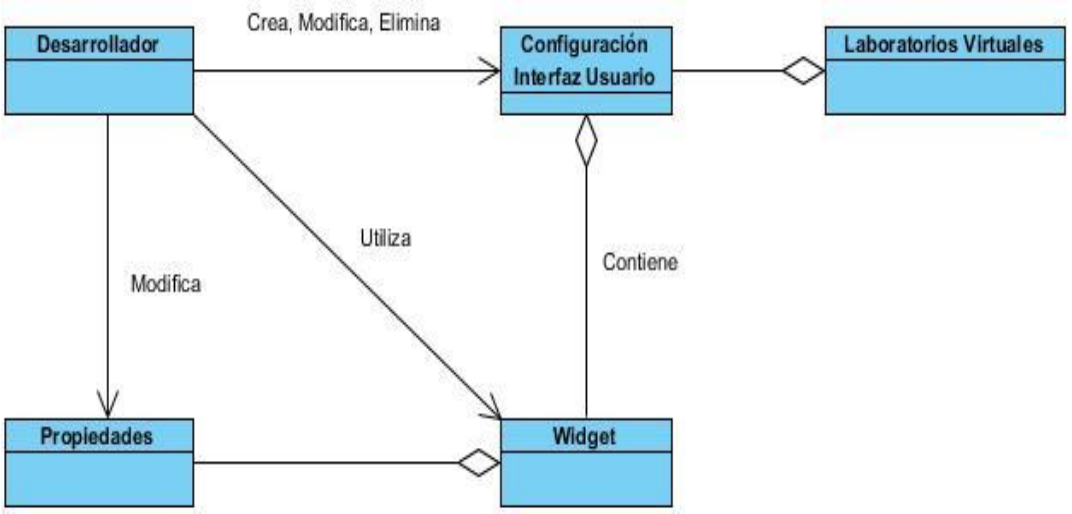

Figura 5: Modelo del dominio

En el mismo se pueden apreciar las representaciones conceptuales que a continuación se describen.

Desarrollador: Es aquel usuario que necesita crear una interfaz gráfica de usuario, el mismo interactuará con el ambiente de trabajo para crear los diseños que desee.

***Widget*:** Son los componentes de una interfaz de usuario, no son más que una abstracción de la realidad, una metáfora para hacer más fácil la experiencia del usuario con el ordenador.

Propiedades: Cada *widget* posee propiedades que la hacen única, en general para cada especie existe un conjunto predefinido de las mismas, que toman valor al instanciarse el individuo, estos valores son los que le interesa al usuario modificar para ajustarlas a sus necesidades.

Configuración: Es un diseño de una interfaz de usuario determinada, con las relaciones que guardan las *widgets* que lo componen entre si, además de las características individuales de cada una.

Laboratorios Virtuales: Es el software que necesita de una interfaz gráfica de usuario para interactuar con el mismo.

2.5 Levantamiento de requisitos del sistema.

Esta etapa del desarrollo es de vital importancia para el resultado final de la aplicación informática, en la misma queda claro que es exactamente lo que se va a hacer. A partir de lo descrito anteriormente y de la comprensión del negocio obtenida, se generará el lenguaje del proceso de ingeniería, por lo que debe

existir una trazabilidad directa entre lo que se determinó como requerimientos y lo que a continuación se mostrará como requisitos del sistema.

2.5.1 Requisitos funcionales generales.

RF1. Gestionar configuración.

RF1.1 Adicionar *widget* a la configuración.

RF1.2 Eliminar *widget* de la configuración.

RF1.3 Modificar *widget* de la configuración.

RF2. Mostrar propiedades de *widgets*.

RF3. Modificar propiedades de *widgets*.

RF4. Mostrar vista previa.

RF5. Exportar configuración.

RF6. Importar configuración.

RF7. Cargar configuración.

2.5.1 Requisitos no funcionales.

Los requisitos no funcionales no juzgan el comportamiento específico de un sistema, sino que especifican los criterios que pueden usarse para juzgar la operación del mismo. A continuación se enumeran estos, por su clasificación.

1. Hardware.

a) Procesador Pentium 4 a 2.7 GHz o superior.
b) Memoria RAM de 1GB de capacidad.
c) Espacio en disco duro de 512 MB.

2. Usabilidad.

a) El sistema deberá ser multiplataforma, es decir deberá correr sobre los sistemas operativos Windows y Linux.

3. Requisitos no funcionales de Restricciones en el Diseño e Implementación.

a) La aplicación se desarrollará utilizando Ogre3D como motor de *render*.
b) Se utilizará QT Creator como IDE de desarrollo.

4. Requisitos no funcionales de Interfaz.

Interfaz de Usuario:

a) Debe permitir realizar todas las operaciones necesarias de forma intuitiva.

2.6 Definición del actor del sistema.

Actor	Descripción
Diseñador	El actor Diseñador es el que realiza la configuración de la interfaz gráfica utilizando el editor, tiene acceso a todas sus funcionalidades.
Programador	El actor Programador es el usuario de la biblioteca de componentes gráficos, por lo tanto es el que carga las configuraciones creadas previamente con el editor utilizando el módulo de carga.

Tabla 1: Actores del sistema

2.7 Casos de uso del sistema.

Los casos de uso son artefactos narrativos que describen el comportamiento de una aplicación teniendo en cuenta el punto de vista del actor, en el Proceso Unificado de Racional, los mismos constituyen el mecanismo rector del proceso de ingeniería, y son una consecuencia directa de los requisitos obtenidos. Intuitivamente muestran la manera en que los usuarios interactúan con el sistema.

2.7.1 Diagrama de paquetes.

El diagrama de paquetes es un mecanismo general para representar de forma organizada los diferentes elementos a ser modelados.

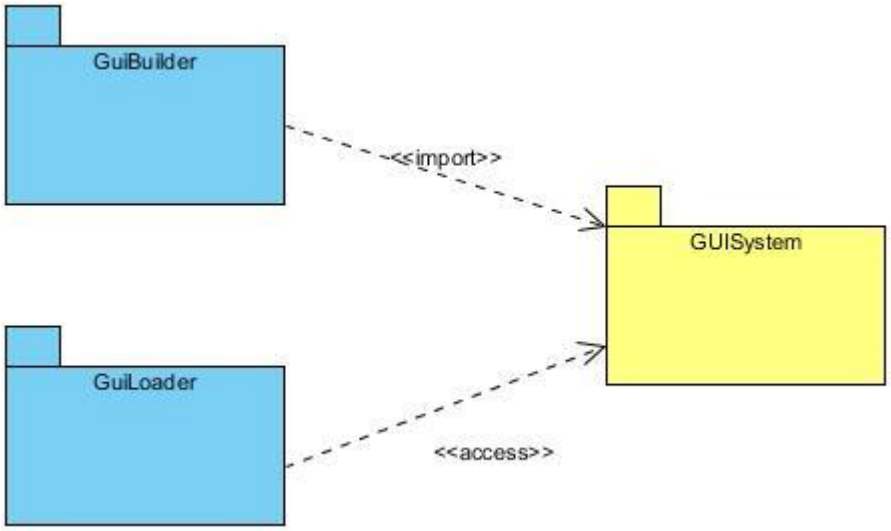

Figura 6: Diagrama de paquetes del sistema

2.7.2 Diagrama de casos de uso del sistema

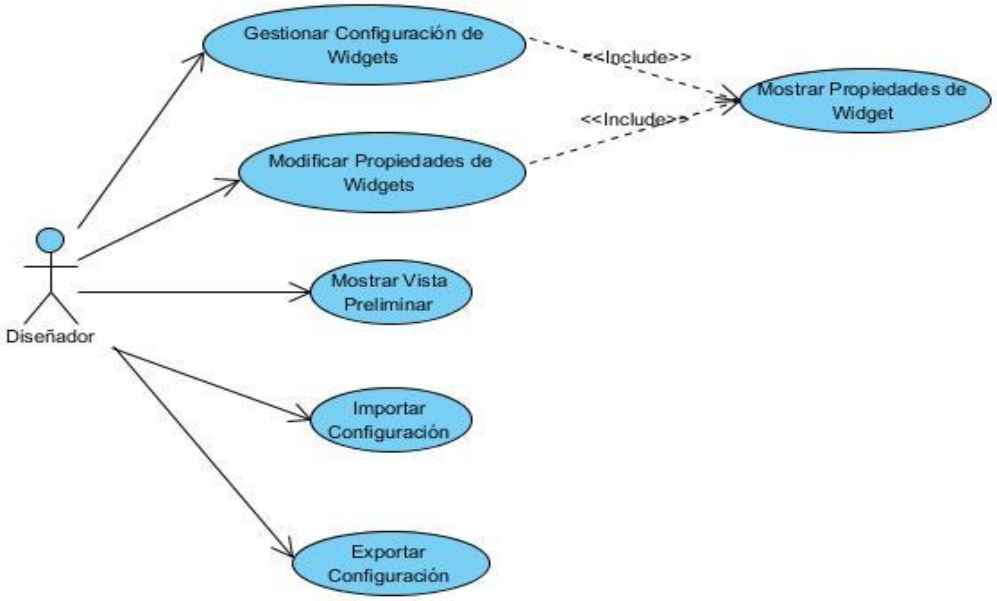

Figura 7: Diagrama de casos de uso del paquete GuiBuilder

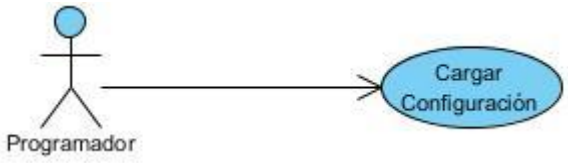

Figura 8: Diagrama de casos de uso del paquete GuiLoader

2.7.3 Descripción de los casos de uso del sistema.

Caso de Uso:	Gestionar Configuración de *Widgets*
Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador necesita gestionar la configuración de los *widgets*. Le permite adicionar, modificar, eliminar o anidar las *widgets* en la configuración.
Precondiciones:	• El sistema debe estarse ejecutando correctamente y todos los recursos deben haber sido previamente cargados.
Referencias	RF1
Reglas del Negocio	• Para que un *widget* permita anidar a otro en su interior este debe ser de tipo Contenedor.
Prioridad	Crítico
Complejidad	Complejo

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador selecciona una de las siguientes opciones: • Adicionar *Widget*, para esto realiza un clic sobre el menú de *widget* al lado izquierdo, encima del *widget* deseado(una vez que haya seleccionado la acción, ver sección	

"Adicionar *Widget*") • Modificar *Widget*, para esto selecciona con el mouse el *widget* que desea(una vez que haya seleccionado la acción, ver sección "Modificar *Widget*") • Eliminar *Widget*, para esto selecciona con el mouse el *widget* que desea(una vez que haya seleccionado la acción, ver sección "Eliminar *Widget*")	
Sección "Adicionar *Widget*"	
Acción del Actor	**Respuesta del Sistema**
	2. El sistema muestra el *widget* seleccionado justo debajo del mouse y crea sus respectivos manejadores de dimensión mientras no se suelte el clic izquierdo.
3. El actor Diseñador mueve el mouse hasta el punto del área de trabajo que desee.	4. El sistema mueve el *widget* siempre debajo de mouse.
5. El actor Diseñador suelta el clic izquierdo sobre un área donde no se encuentre ningún *widget* contenedor.	6. El sistema muestra el *widget* en el punto dado, con sus manejadores de dimensión. 7. Invoca el caso de uso Mostrar Propiedades de *Widget*. 8. Termina el caso de uso.
Sección "Modificar *Widget*"	

Acción del Actor	Respuesta del Sistema
	2. El sistema muestra los manejadores de dimensión del *widget* seleccionado.
3. El actor Diseñador coloca el puntero del mouse sobre alguno de los manejadores de dimensión del *widget*.	4. El sistema cambia el puntero del mouse al modo de redimensionar.
5. El actor Diseñador presiona el clic izquierdo y mueve el mouse para redimensionar en la dirección que desee.	6. El sistema cambia el tamaño del *widget* de acuerdo al movimiento del puntero.
7. El actor Diseñador suelta el clic izquierdo para indicar que obtuvo el tamaño deseado	8. El sistema invoca el caso de uso Mostrar Propiedades de *Widget*. 9. Termina el caso de uso.
Flujo alterno al paso 2 "Seleccionar más *Widgets*"	
Acción del Actor	**Respuesta del Sistema**
3.1 El actor Diseñador presiona las teclas Ctrl o Shift y presiona a la vez el clic izquierdo del mouse sobre una *widget* no seleccionada.	3.2 El sistema muestra los manejadores de dimensión del nuevo *widget* seleccionado
3.3 El actor Diseñador presiona las teclas Ctrl o Shift y presiona a la vez el clic izquierdo del mouse sobre una *widget* no seleccionada.	3.4 Realizar paso 3.2.
Flujo alterno al paso 2 "Mover *Widget*"	
Acción del Actor	**Respuesta del Sistema**
2.1 El actor Diseñador presiona clic izquierdo sobre alguna de las *widgets* seleccionadas y arrastra el mouse hasta la posición en que desea colocarlas.	2.2 El sistema mueve la *widget* hasta el lugar indicado.

2.3 El actor Diseñador suelta el clic izquierdo.	2.4 El sistema verifica que el mouse se encuentre sobre una *widget* de tipo contenedor, y de ser así adjunta las *widgets* como hijas del contenedor. 2.5 Si es solamente una *widget* invoca el caso de uso Mostrar Propiedades de *Widget*. 2.6 Termina el caso de uso.
Flujo alterno al paso 2 "Deseleccionar *Widget*"	
Acción del Actor	**Respuesta del Sistema**
2.1 El actor Diseñador presiona clic izquierdo en un área en que no se encuentre ningún *widget*.	2.2 El sistema elimina los manejadores de dimensión de los *widgets* que estaban seleccionados. 2.3 Termina el caso de uso.
Sección "Eliminar *Widget*"	
Acción del Actor	**Respuesta del Sistema**
	2. El sistema muestra los manejadores de dimensión del *widget* seleccionado.
3. El actor Diseñador presiona la tecla Delete.	4. El sistema elimina el *widget* seleccionado. 5. Termina el caso de uso.
Flujo alterno al paso 2 "Seleccionar más *Widget*"	
Acción del Actor	**Respuesta del Sistema**
2.1 El actor Diseñador presiona las teclas Ctrl o Shift y presiona a la vez el clic izquierdo del mouse sobre una *widget* no seleccionada.	2.2 El sistema muestra los manejadores de dimensión del *widget* agregado a los seleccionados.

2.3 El actor Diseñador presiona las teclas Ctrl o Shift y presiona a la vez el clic izquierdo del mouse sobre una *widget* no seleccionada.	2.4 Realizar paso 2.2 mientras se repita la acción, en caso contrario ir al paso 2 de la sección.
Flujo alterno al paso 2 "Deseleccionar *Widgets*"	
Acción del Actor	**Respuesta del Sistema**
2.1 El actor Diseñador presiona clic izquierdo en un área en que no se encuentre ningún *widget*.	2.2 El sistema elimina los manejadores de dimensión de los *widgets* que estaban seleccionados. 2.3 Termina el caso de uso.
Pos-condiciones	• El sistema queda con la configuración deseada por el usuario.

Tabla 2: CUS Gestionar configuración.

Caso de Uso:	Mostrar Propiedades de *Widget*
Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador necesita ver los valores que poseen las propiedades de un *widget*.
Precondiciones:	• El *widget* debe estar visible y en la configuración
Referencias	RF2
Reglas del Negocio	• Solo se mostrará las propiedades de un *widget* a la vez.
Prioridad	Crítico
Complejidad	Baja
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador presiona el clic izquierdo sobre la *widget* que desea obtener las propiedades y luego lo libera.	2. El sistema muestra los manejadores de dimensión de la *widget* seleccionada. 3. El sistema muestra en un menú las propiedades del *widget*.

	4. Termina el caso de uso
Pos-condiciones	• Las propiedades del *widget* son mostradas.

Tabla 3: CUS Mostrar propiedades de *widget*

Caso de Uso:	Modificar Propiedades de *Widget*
Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador necesita modificar las propiedades de un *widget* con mayor exactitud.
Precondiciones:	• El *widget* debe estar adicionado a la configuración.
Referencias	RF3
Reglas del Negocio	• Dos *widgets* no pueden tener el mismo nombre. • Los recursos como materiales y tipos de fuente que se vayan a usar deben haber sido cargados previamente.
Prioridad	Crítico
Complejidad	Media
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador realiza un clic sobre la *widget* que desea modificar las propiedades.	2. El sistema muestra los manejadores de dimensión de la *widget* seleccionada. 3. El sistema muestra en un menú las propiedades del *widget*.
4. El actor diseñador realiza un clic sobre el valor de la propiedad que desee modificar.	5. El sistema cambia el color del texto de la propiedad para indicar que puede ser editada
6. El actor modifica la propiedad ingresando un valor mediante el teclado y presiona la tecla Enter para indicar que terminó dicha modificación.	7. El sistema comprueba la validez de los datos. 8. El sistema modifica la propiedad según el dato entrado y modifica el *widget*. 9. Termina el caso de uso.

Flujo alterno al paso 4 "Deseleccionar *Widget*"	
Acción del Actor	**Respuesta del Sistema**
4.1 El actor Diseñador presiona clic izquierdo en un área en que no se encuentre ningún *widget*.	4.2 El sistema elimina los manejadores de dimensión de los *widgets* que estaban seleccionados. 4.3 Termina el caso de uso.
Flujo alterno al paso 7 "Datos Incorrectos"	
Acción del Actor	**Respuesta del Sistema**
	7.1 El sistema muestra un mensaje indicando que los datos ingresados no están en el formato adecuado.
Flujo alterno al paso 7 "Recurso no Cargado"	
Acción del Actor	**Respuesta del Sistema**
	7.1 El sistema solicita ingresar la ruta de los recursos necesarios.
7.2 El actor Diseñador selecciona la dirección de los recursos.	7.3 El sistema carga los recursos. 7.4 El sistema modifica la propiedad según el dato entrado y modifica el *widget*. 7.5 Termina el caso de uso.
Flujo alterno al paso 7.2 "No selecciona dirección"	
Acción del Actor	**Respuesta del Sistema**
7.2.1 Selecciona la opción de cancelar.	7.2.2 El sistema no modifica la propiedad. 7.2.3 Termina el caso de uso
Pos-condiciones	• Las propiedades del *widget* fueron cambiadas y los cambios aplicados.

Tabla 4: CUS Modificar propiedades de *widget*

Caso de Uso:	Mostrar Vista Preliminar

Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador desea ver cómo quedará el diseño cuando esté corriendo en la aplicación.
Precondiciones:	• Debe haber una configuración realizada.
Referencias	RF4
Reglas del Negocio	-
Prioridad	Mediana
Complejidad	Baja

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador desea tener una vista previa y sigue la ruta de la barra de menú "View/Show preview"	2. El sistema muestra la vista previa.
3. El actor interactúa con los *widgets* de la configuración según desee.	4. El sistema responde a los eventos lanzados por el actor sobre las *widgets*.
5. El actor termina la vista previa siguiendo la ruta "View/End preview".	6. El sistema termina la vista previa y vuelve al estado en que se encontraba anteriormente. 7. Termina el caso de uso.
Pos-condiciones	• El sistema queda exactamente igual al momento antes de mostrar la vista previa.

Tabla 5: CUS Mostrar vista preliminar

Caso de Uso:	Exportar Configuración
Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador desea exportar una configuración realizada.

Precondiciones:	• Debe haber una configuración realizada.
Referencias	RF5
Reglas del Negocio	-
Prioridad	Crítico
Complejidad	Media
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador desea exportar una configuración y sigue la ruta de la barra de menú "File/Save" o presiona simultáneamente las teclas Ctrl + S.	2. El sistema solicita que ingrese la ruta y nombre del archivo en que se guardará la configuración.
3. El actor selecciona la ruta e ingresa un nombre de archivo.	4. El sistema comprueba la validez del nombre y la ruta ingresada. 5. El sistema crea el archivo con el nombre y la ruta ingresada, y guarda en este la configuración. 6. Termina el caso de uso.
Flujo alterno al paso 3 "Cancelar Exportación"	
Acción del Actor	**Respuesta del Sistema**
3.1 El actor Diseñador presiona el botón cancelar.	3.2 Termina el caso de uso.
Pos-condiciones	• La configuración es guardada en un archivo.

Tabla 6: CUS Exportar configuración

Caso de Uso:	Importar Configuración
Actores:	Diseñador
Resumen:	El caso de uso se inicia cuando el Diseñador desea importar una

	configuración previamente exportada.
Precondiciones:	-
Referencias	RF6
Reglas del Negocio	-
Prioridad	Crítico
Complejidad	Media

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Diseñador desea importar una configuración y sigue la ruta de la barra de menú “File/Load” o presiona simultáneamente las teclas Ctrl + O.	2. El sistema solicita que ingrese la ruta y nombre del archivo del que se cargará la configuración.
3. El actor selecciona la ruta y un archivo en esta.	4. El sistema comprueba la validez del nombre y la ruta ingresada. 5. El sistema comprueba que el archivo sea válido 6. El sistema muestra la configuración del archivo. 7. Termina el caso de uso.
Flujo alterno al paso 3 “Cancelar Importación”	
Acción del Actor	**Respuesta del Sistema**
2.1 El actor Diseñador presiona el botón cancelar.	2.2 Termina el caso de uso.
Flujo alterno al paso 5 “Archivo no Válido”	
Acción del Actor	**Respuesta del Sistema**
	3.1 El sistema muestra un mensaje informando que el archivo no cumple con

	el formato adecuado. 3.2 Termina el caso de uso.
Pos-condiciones	• La configuración es importada en un archivo.

Tabla 7: CUS Importar configuración

Caso de Uso:	Cargar Configuración
Actores:	Programador
Resumen:	El caso de uso se inicia cuando el Programador desea cargar una configuración previamente exportada.
Precondiciones:	-
Referencias	RF7
Reglas del Negocio	• El programador debe contar con la biblioteca de componentes gráficos y tener un objeto del tipo GuiManager creado.
Prioridad	Crítico
Complejidad	Media
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor Programador desea cargar una configuración para esto utilizando una instancia de GuiManager invoca el método cargarConfiguracion() el cual recibe como parámetro la dirección y el nombre del archivo de configuración	2. Durante la ejecución el sistema comprueba la validez del nombre y la ruta ingresada. 3. El sistema comprueba que el archivo sea válido 4. El sistema carga la configuración 5. Termina el caso de uso
Flujo alterno al paso 2 "Ruta o Nombre no Válido"	
Acción del Actor	**Respuesta del Sistema**
	2.1 El sistema lanza una excepción informando que el archivo es inválido.

	2.2 Termina el caso de uso.
Flujo alterno al paso 3 “Archivo no Válido”	
Acción del Actor	**Respuesta del Sistema**
	3.1 El sistema lanza una excepción informando que el archivo no cumple con el formato adecuado. 3.2 Termina el caso de uso.
Pos-condiciones	• La configuración es cargada en una aplicación.

Tabla 8: CUS Cargar configuración

Conclusiones generales del capítulo.

La descripción de la solución que se propone al problema de la investigación, en conjunto con la descripción del modelo del dominio y las especificaciones de los casos de uso, proporcionan una entrada para los flujos de trabajo de la ingeniería que se reflejan en los siguientes capítulos. Por otra parte la descripción de los casos de uso permite determinar que funcionalidades deben ser priorizadas a continuación.

CAPÍTULO 3: DISEÑO DE LA SOLUCIÓN

Introducción.

En este capítulo se abordará el proceso de diseño de la solución, para eso se utilizarán artefactos conocidos como los diagramas de clase del diseño que presenta las clases del sistema con sus relaciones estructurales y de herencia, y los diagramas de secuencia que son usados para describir gráficamente un caso de uso o un escenario.

3.1 Diseño del sistema.

El diseño del sistema surge con la necesidad de ubicar en el contexto del lenguaje de programación, sistemas operativos y tecnologías de interfaz de usuario, los casos de uso obtenidos en fases previas del proceso de desarrollo, para alcanzar una mejor comprensión.

Este flujo es el encargado de proporcionar una entrada indispensable en la etapa posterior de implementación.

Constituyen artefactos vitales del mismo los diagramas de clases de diseño, y los diagramas de interacción de los cuales existen dos variantes, los de secuencia y los de colaboración.

3.2 Diagrama de clases del diseño.

Es el principal diagrama para el diseño del sistema, representa las relaciones estructurales y de herencia de las clases, además muestra de manera descriptiva los atributos que poseen y las relaciones entre estas últimas.

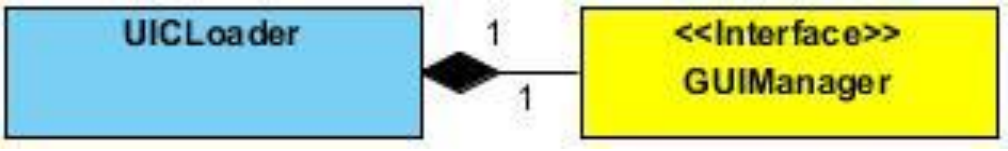

Figura 9: Diagrama de clases del diseño del paquete GuiLoader

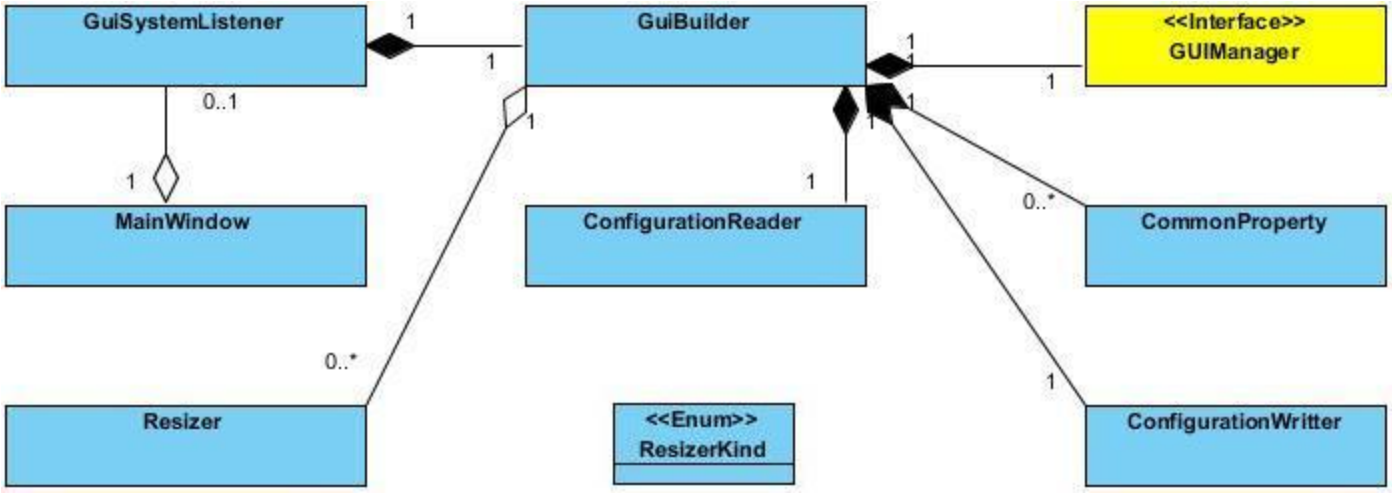

Figura 10: Diagrama de clases del diseño del paquete GuiBuilder

3.3 Descripción de las clases.

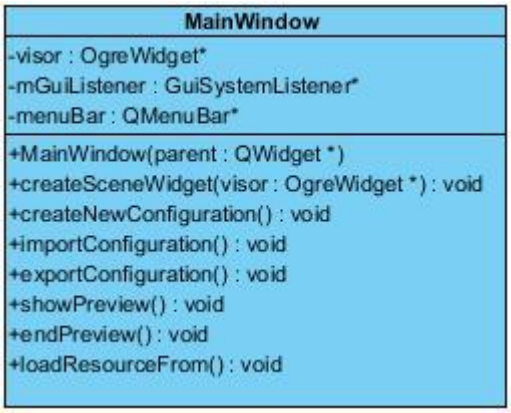

Figura 11: Clase MainWindow

Descripción: Constituye la ventana principal de la aplicación, contiene además de la barra de menú, un *widget* especial llamado visor, que es a su vez la ventana donde visualiza Ogre3D.

Figura 12: Clase Resizer

Descripción: Es una clase que representa los manejadores de dimensión, y su correspondiente objeto en Ogre3D.

Figura 13: Clase GuiSystemListener

Descripción: Constituye la interfaz principal con Ogre3D, la misma brinda acceso a los métodos del ciclo de *render*, y permite manejar los eventos.

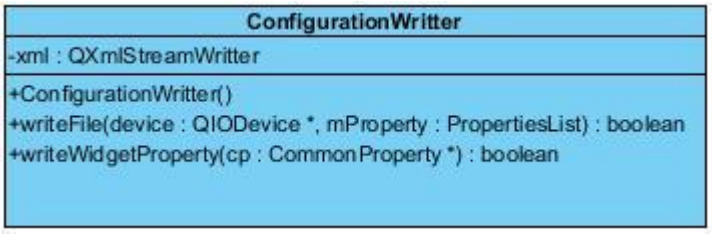

Figura 14: Clase ConfigurationWritter

Descripción: Es la clase que permite exportar las configuraciones, a partir de una lista de propiedades genera un archivo con esta descripción.

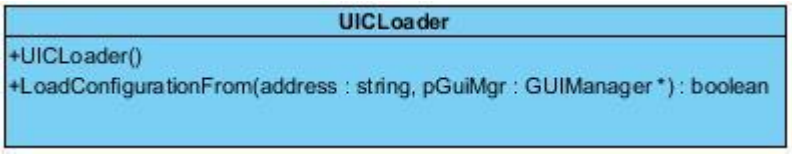

Figura 15: Clase UICLoader

Descripción: Es la clase que permite cargar en una aplicación de Ogre3D, una configuración previamente realizada en el GuiBuilder.

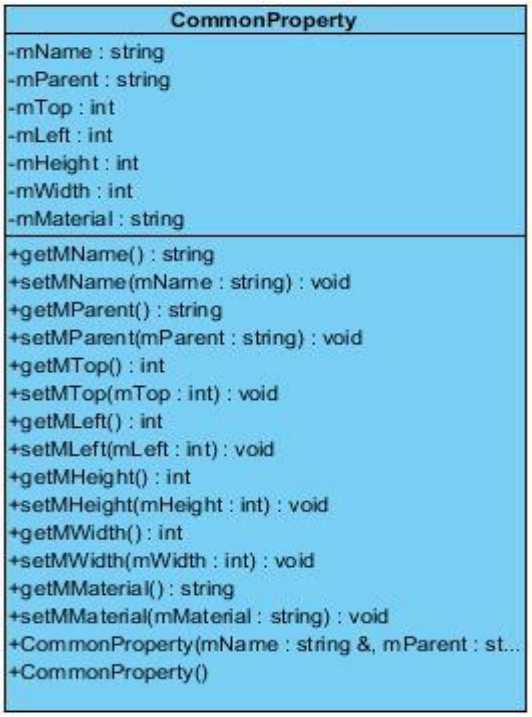

Figura 16: Clase CommonProperty

Descripción: Es una clase entidad que almacena temporalmente los datos de los *widgets* y brinda acceso a los mismos.

Figura 17: Clase GuiBuilder

Descripción: Constituye la clase controladora de la aplicación, implementa la lógica de las funcionalidades de la misma.

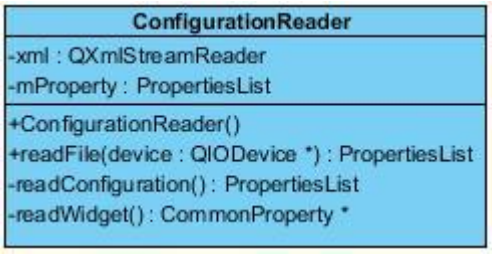

Figura 18: Clase ConfigurationReader

Descripción: Es la clase que permite importar las configuraciones, a partir de un archivo con la descripción necesaria.

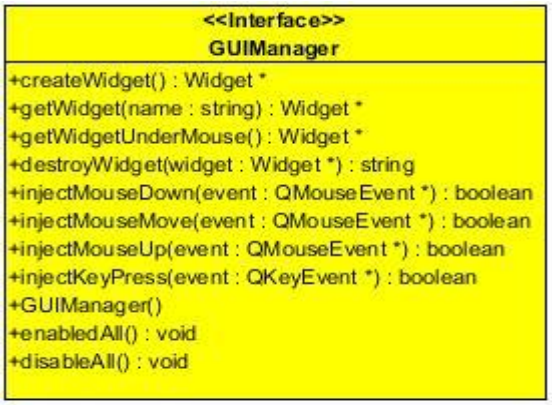

Figura 19: Clase GUIManager

Descripción: Es la clase interfaz de la biblioteca GuiSystem, no pertenece a ninguno de los dos paquetes, pero se muestra su descripción para una mayor claridad en el diseño.

3.4 Diagramas de interacción.

Los diagramas de interacción describen el comportamiento de los objetos dentro de un caso de uso o sección.

Un diagrama de secuencia muestra las interacciones entre objetos según un punto de vista temporal, por lo que representa dicha interacción en base a la cronología de los envíos de mensajes.

En este se pueden apreciar líneas verticales que representan los objetos de un escenario y los mensajes entre dichos objetos son flechas que los conectan. Los períodos de actividad de estos objetos están representados por los rectángulos a lo largo de las líneas verticales. En estos diagramas los actores son encargados de generar varios eventos.

A continuación se muestran los diagramas de interacción para los diferentes escenarios de ambos paquetes.

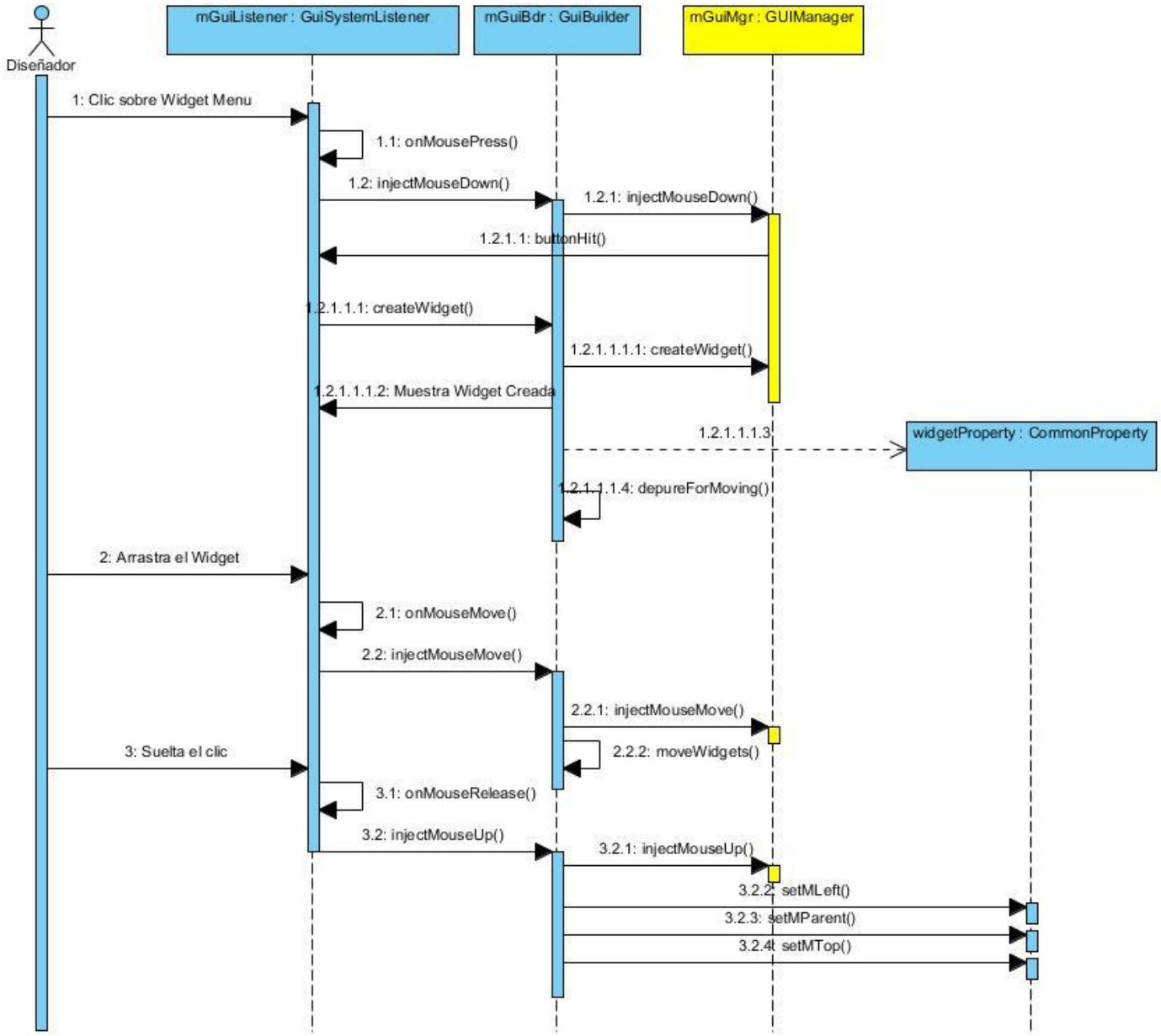

Figura 20: Diagrama de secuencia adicionar *widget*

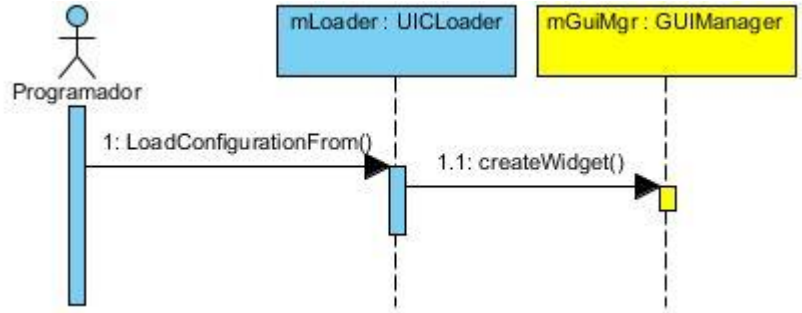

Figura 21: Diagrama de secuencia cargar configuración

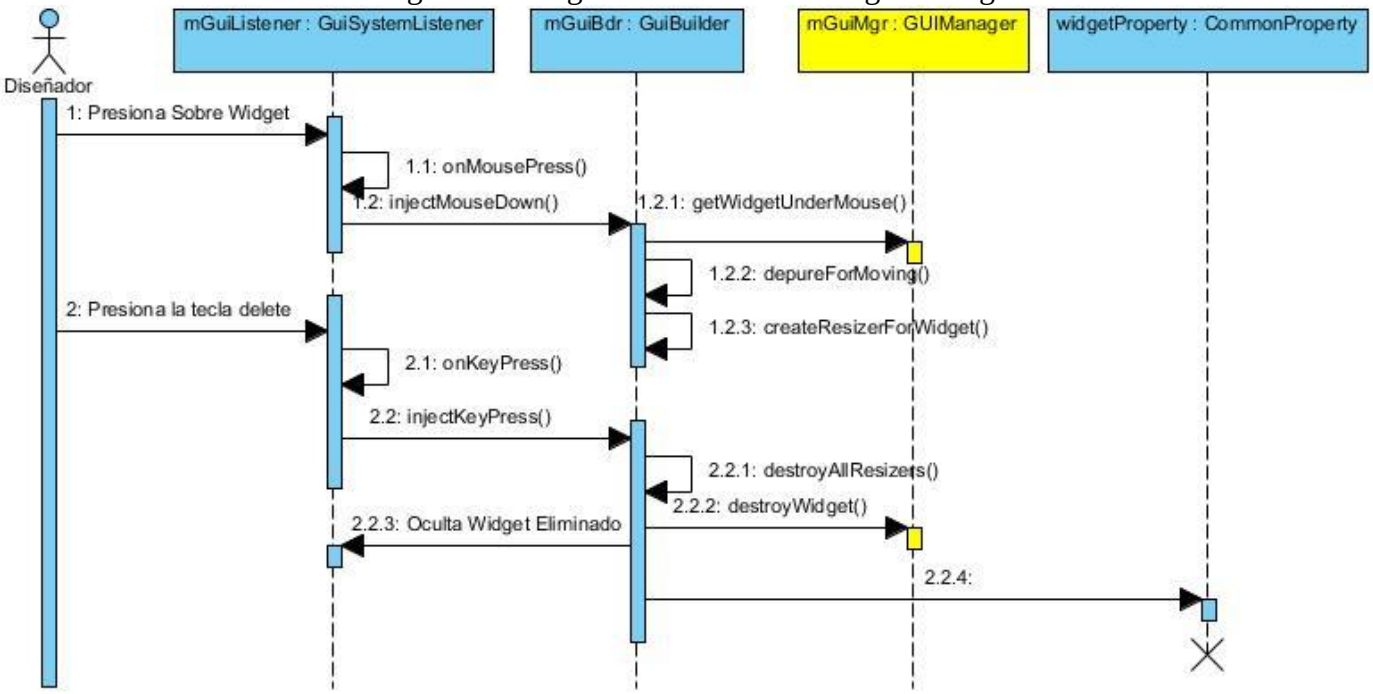

Figura 22: Diagrama de secuencia eliminar *widget*

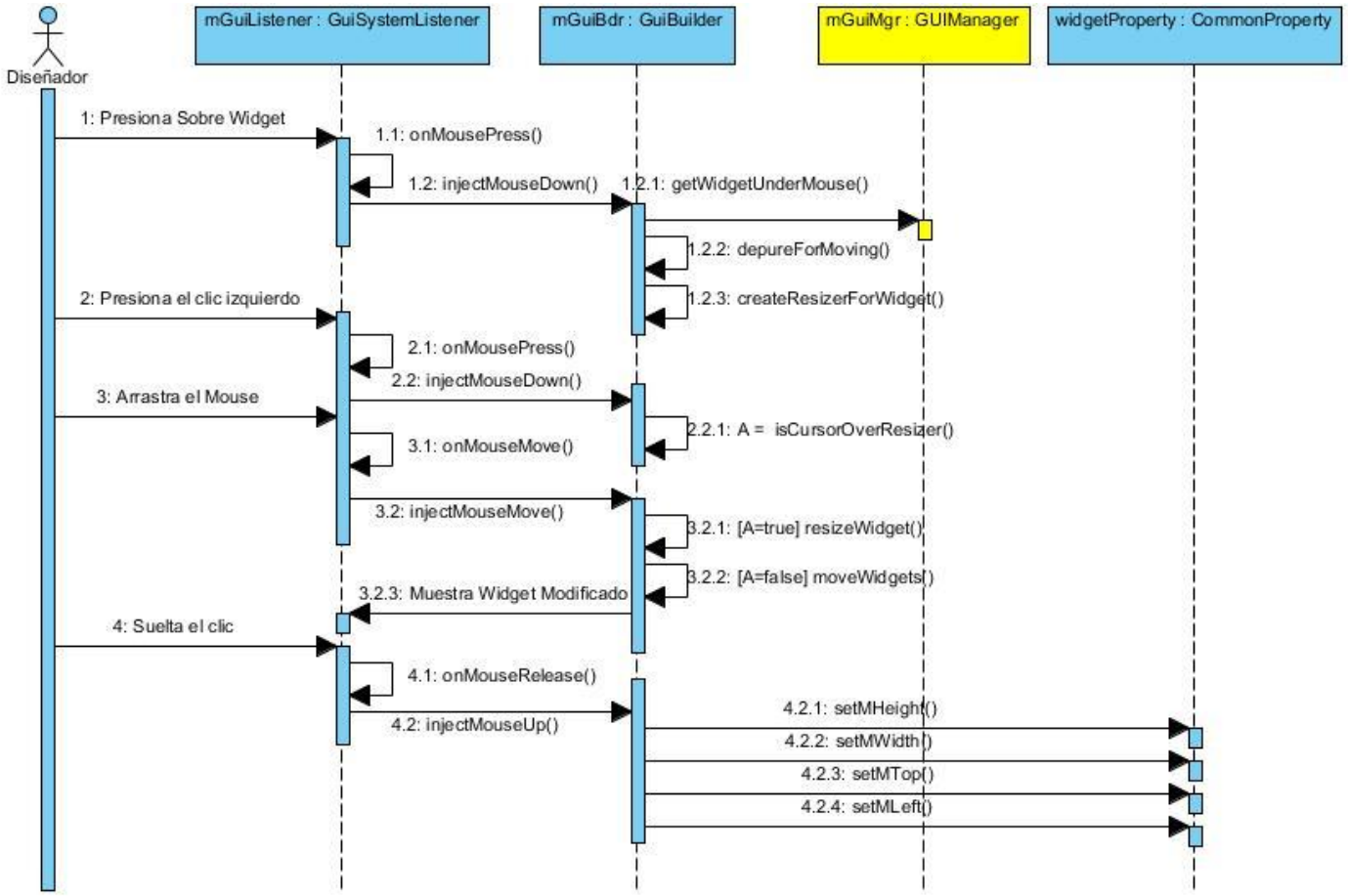

Figura 23: Diagrama de secuencia modificar *widget*

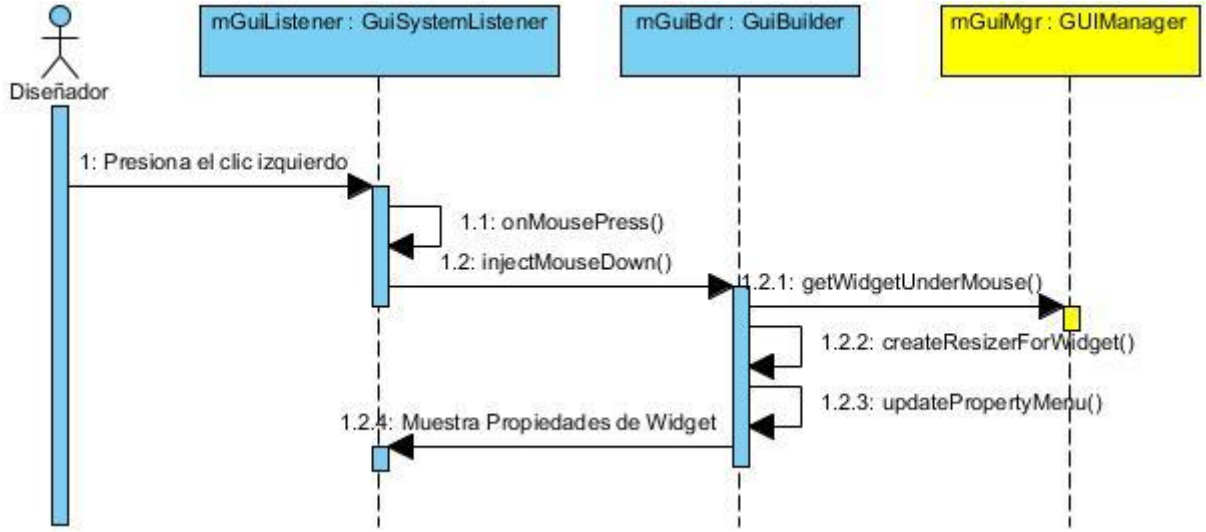

Figura 24: Diagrama de secuencia mostrar propiedades de *widget*

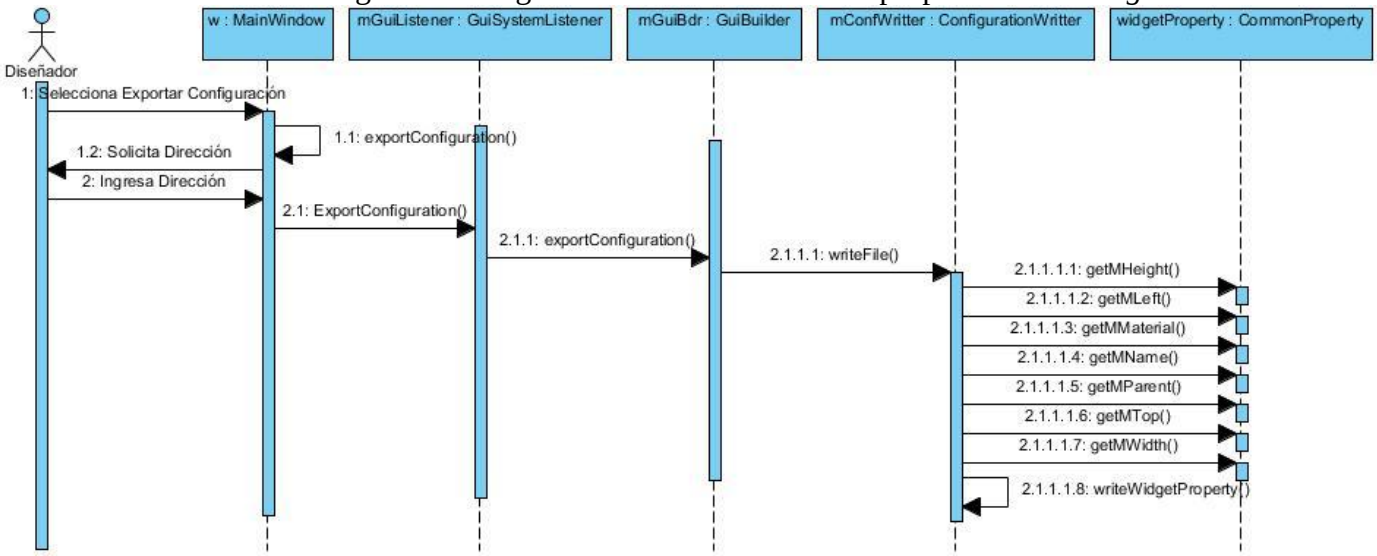

Figura 25: Diagrama de secuencia exportar configuración

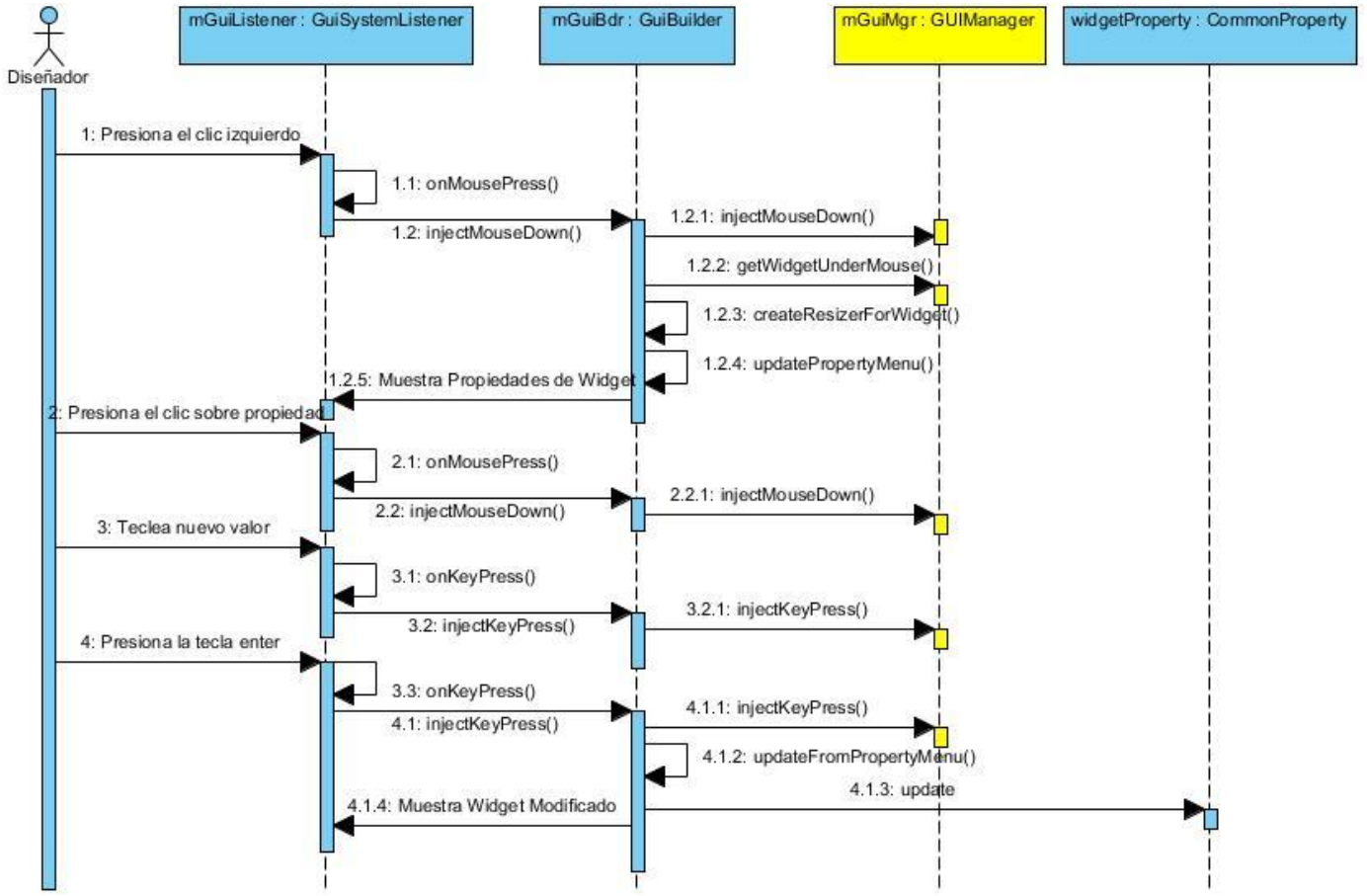

Figura 26: Diagrama de secuencia modificar propiedades de *widget*

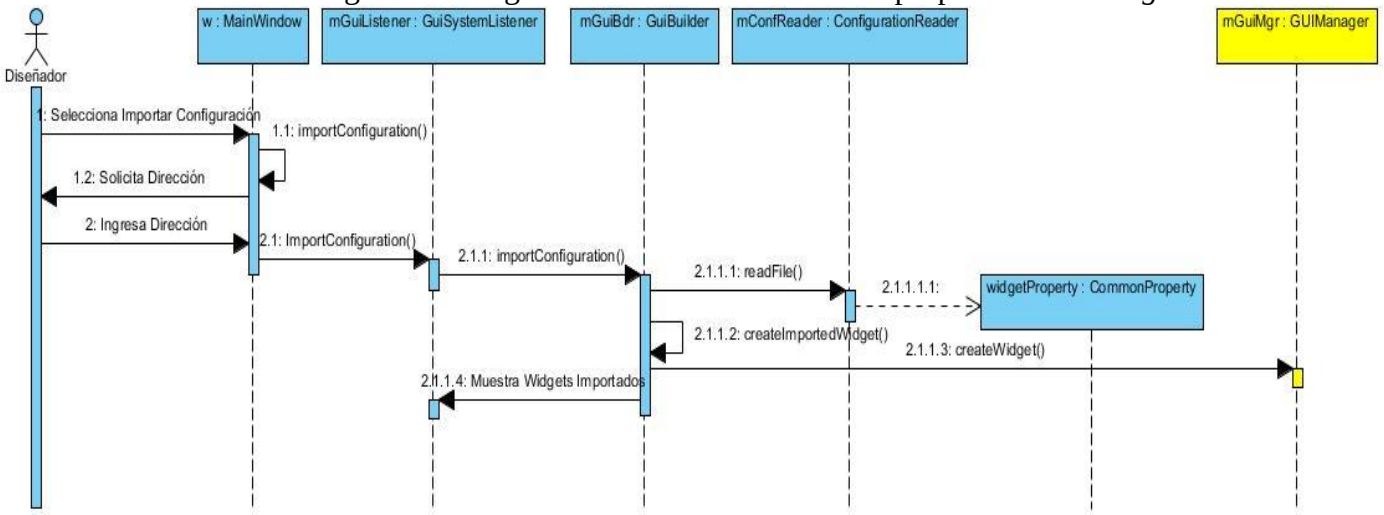

Figura 27: Diagrama de secuencia importar configuración

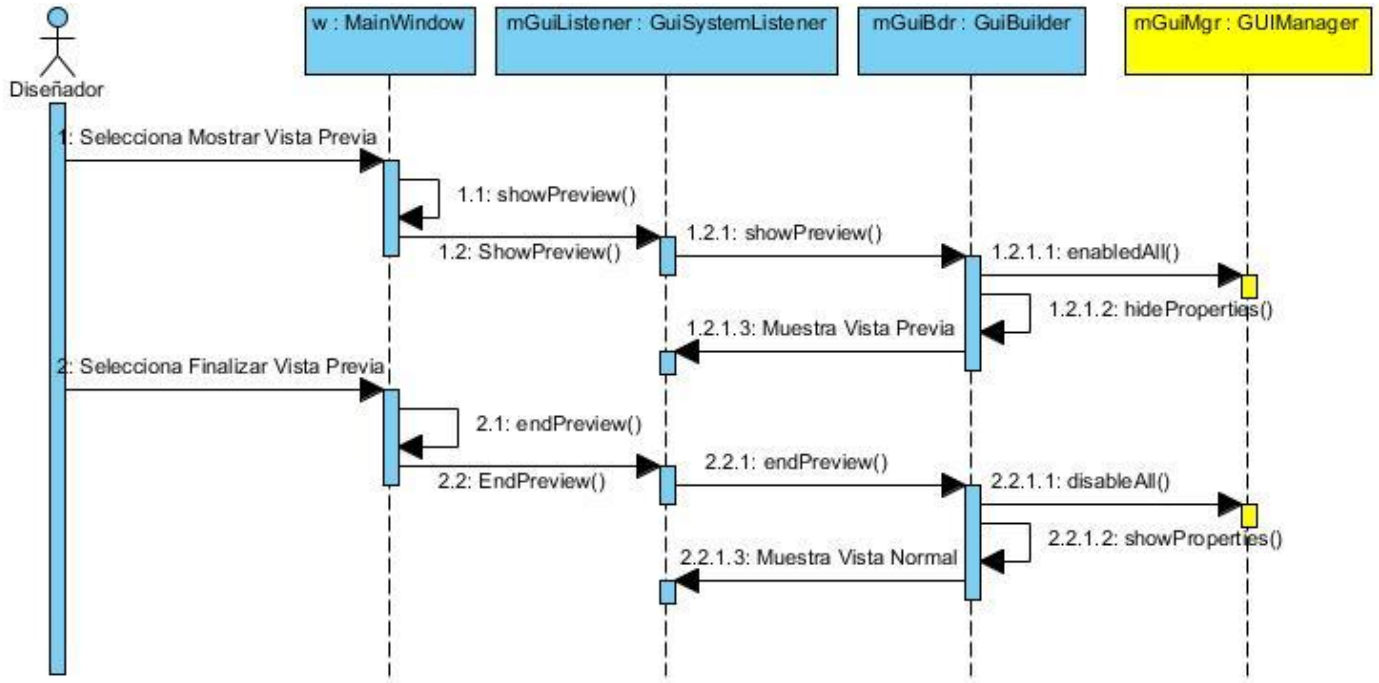

Figura 28: Diagrama de secuencia mostrar vista preliminar

Conclusiones generales del capítulo.

En este capítulo se presentaron artefactos tan importantes como el diagrama de clases del diseño y los diagramas de secuencia para cada escenario posible, lo cual constituye el punto de partida para la siguiente etapa del desarrollo, o sea, la implementación. En este momento se encuentra definida una línea base de la arquitectura.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

Introducción.

La implementación constituye el flujo de la ingeniería que mayor cantidad de recursos involucra, tomando como entrada en cada iteración los elementos obtenidos del diseño, esta se encarga de traducirlos en código fuente y ficheros ejecutables, que inicialmente serán un prototipo del entregable y posteriormente constituirán el producto final. Tras culminado este flujo todos los requisitos, ya sea funcionales o no funcionales, deben haber sido cumplidos, al menos en un prototipo. A este flujo está dedicado el presente capítulo.

4.1 Diagramas de componentes.

Un componente es un grupo de clases que trabajan estrechamente y puede corresponder a código fuente, binario o ejecutable; por lo que los diagramas de estos permiten modelar la estructura del software y las dependencias entre los mismos. En general los componentes dejan de ser una representación simbólica del sistema y se apega a las partes físicas que realmente lo compondrán. La estructura del editor de configuraciones de interfaz gráfica de usuario las podemos ver a continuación.

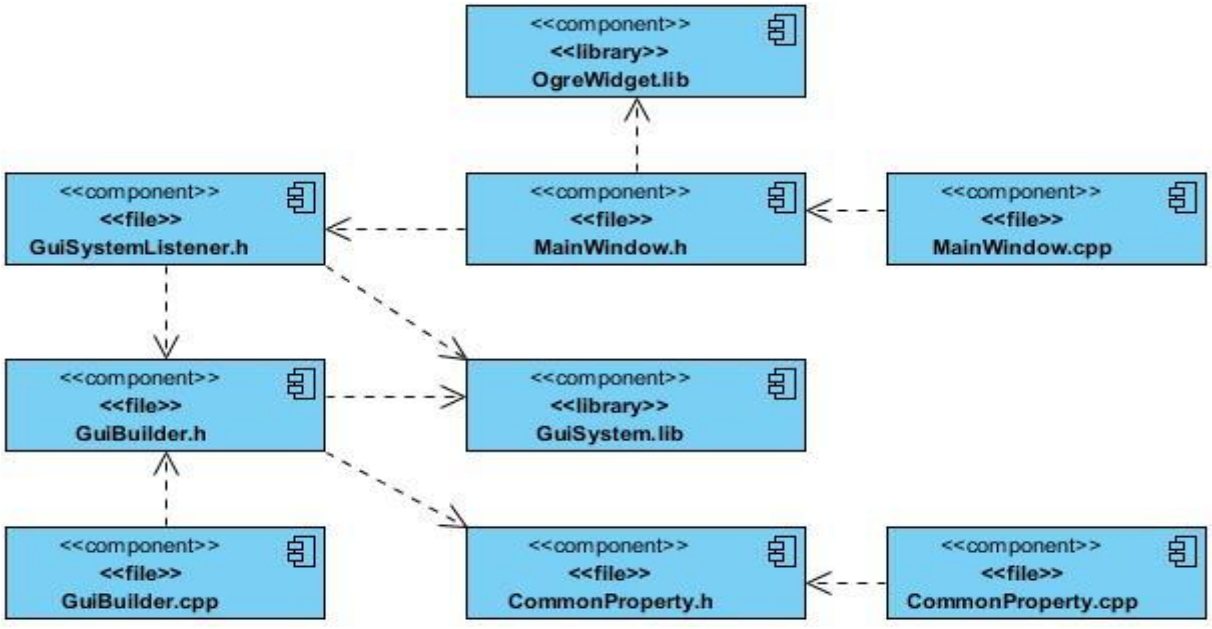

Figura 29: Diagrama de componentes del paquete GuiBuilder

Figura 30: Diagrama de componentes del paquete GuiLoader

4.2 Diagramas de Estado.

Los diagramas de estado se utilizan generalmente para mostrar el comportamiento de los objetos de una determinada clase, los eventos que causan transiciones en esta, y las acciones que resultan desde un cambio de estado; pueden usarse en todas las clases pero son realmente útiles en aquellas en que varíe el comportamiento a partir de determinados eventos.

Por la claridad que brinda para la implementación se definieron los siguientes diagramas de estado para la clase GuiBuilder.

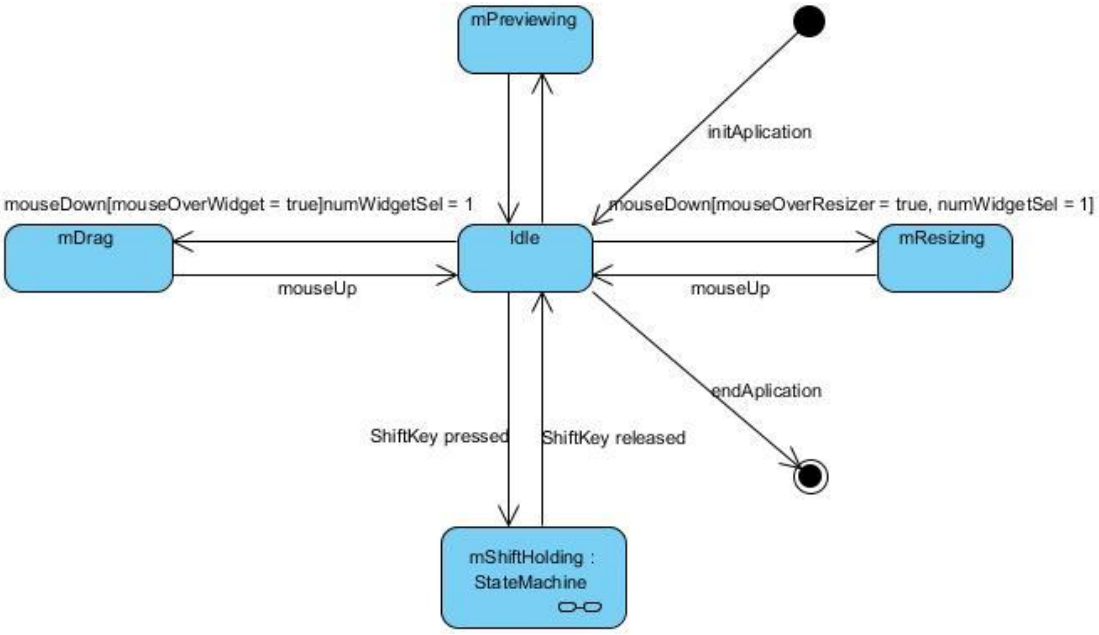

Figura 31: Diagrama de estado para la clase GuiBuilder

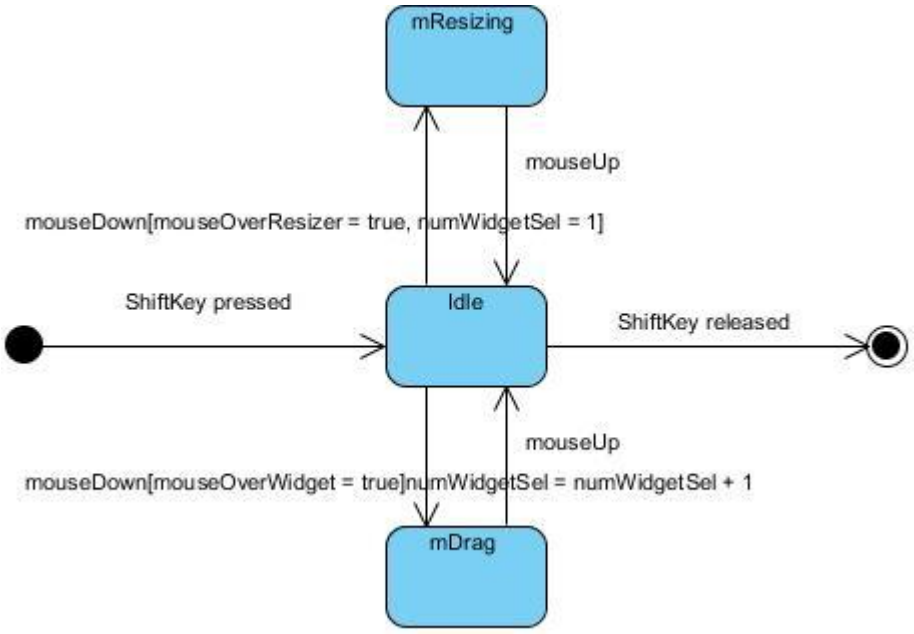

Figura 32: Diagrama de la máquina de estado mShiftHolding

4.3 Validación del software.

Las pruebas realizadas al software constituyen su única garantía de calidad, las mismas pueden estar dirigidas a los componentes de forma individual o al sistema como un conjunto. Siguiendo la línea de la metodología, estas pruebas, están dirigidas por cada caso de uso que se traducen a su vez en casos de prueba.

Las pruebas realizadas son especificadas ubicándolas primero en su sección y escenario, luego se explica la funcionalidad a validar y por último se describen las variables que intervienen en dicha funcionalidad.

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad
SC 1. Adicionar *Widget*	EC 1.1 Adicionar *Widget*	El usuario tendrá la posibilidad de adicionar *widgets* a la configuración arrastrándolos a partir del menú llamado "*Widget*'s

		Menu”
SC 2. Modificar *Widget*	EC 2.1 Modificar Dimensiones de *Widget*	El usuario tendrá la posibilidad de modificar las dimensiones de un *widget* usando las marcas de dimensión que aparecen tras seleccionarlo. Una vez concluido las propiedades deben mostrarse actualizadas.
	EC 2.2 Modificar Posición de *Widget*	El usuario tendrá la posibilidad de cambiar la posición de uno o varios *widgets* a la vez, tras seleccionarlos, con solo arrastrarlos hasta la nueva posición.
	EC 2.3 Seleccionar Varios *Widgets*	El usuario tendrá la posibilidad de seleccionar varios *widgets* a la vez manteniendo oprimida la tecla Shift mientras los selecciona con el clic izquierdo. Mientras haya más de una *widget* seleccionada no se mostrará ninguna propiedad.

	EC 2.4 Deseleccionar *Widget*	El usuario podrá deseleccionar un *widget* previamente seleccionado al hacer clic sobre uno que no esté seleccionado, o sobre un área donde no haya ningún *widget*. También puede deseleccionarlo al hacer clic sobre el *widget* que desee excluir si tiene la tecla Shift oprimida.
SC 3. Eliminar *Widget*	EC 3.1 Eliminar *Widget*	El usuario podrá eliminar los *widgets* que tenga seleccionados oprimiendo la tecla Delete.

Tabla 9: Validación del CU gestionar configuración de *widgets*

Descripción de las variables:

***Widget*:** Cada uno de los elementos que conforman una configuración.

Figura 33: Adicionar *widget*

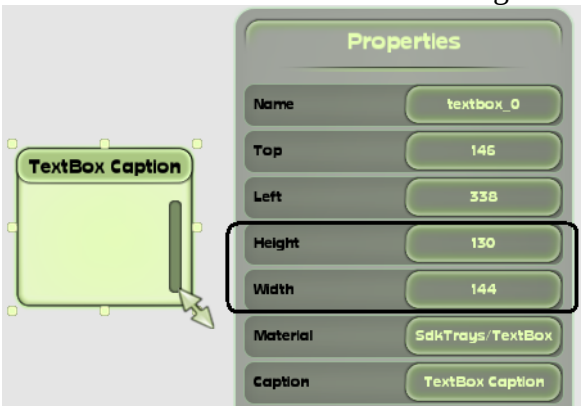

Figura 34: Modificar dimensiones antes

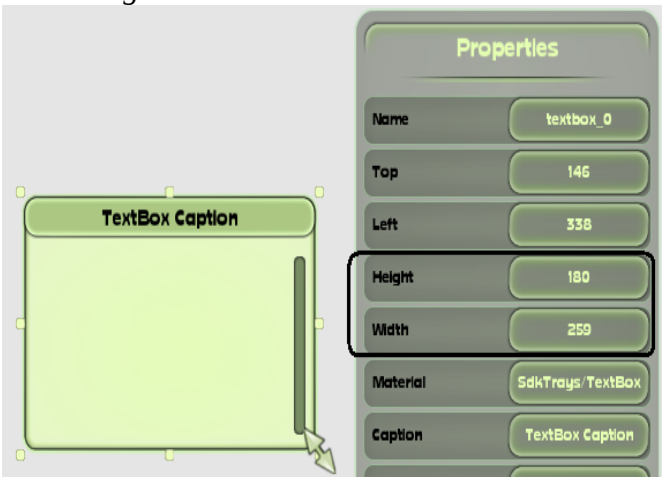

Figura 35: Modificar dimensiones después

Figura 36: Modificar posición antes

Figura 37: Modificar posición después

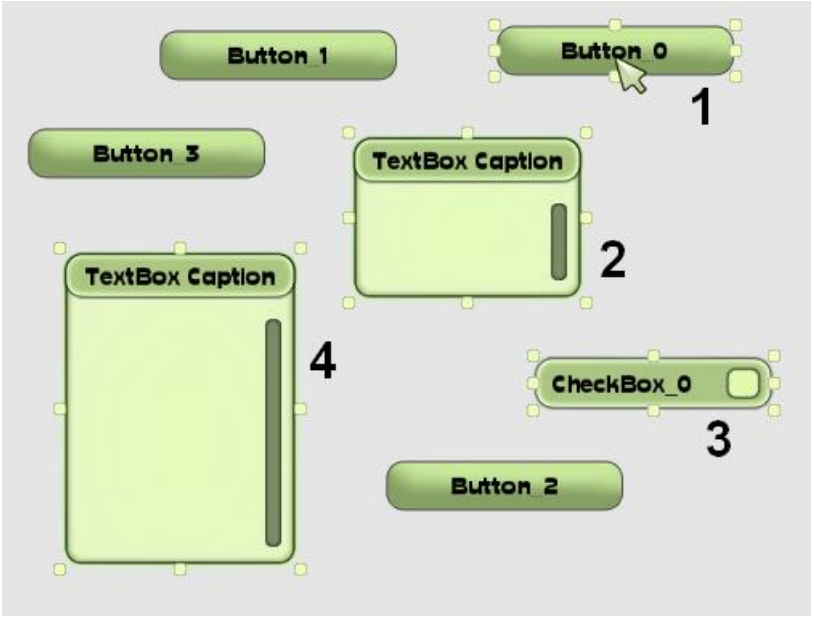

Figura 38: Seleccionar más *widgets*

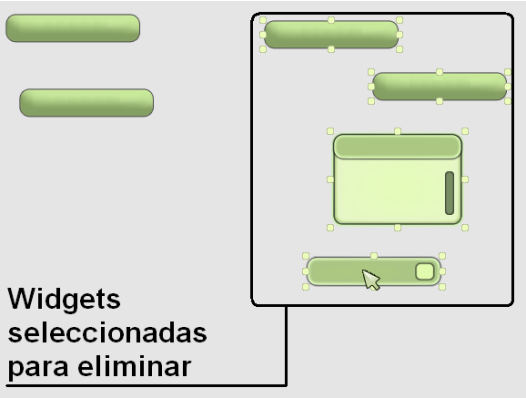

Figura 39: Eliminar *widgets* antes

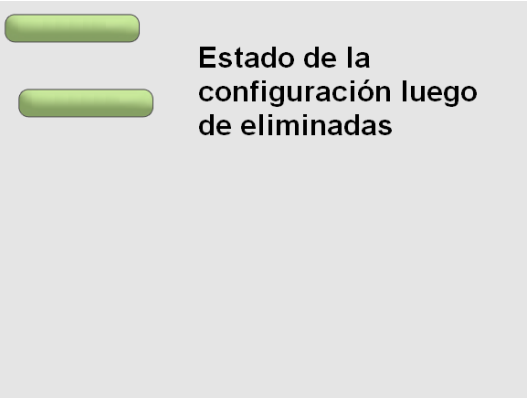

Figura 40: Eliminar *widgets* después

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad
SC 1. Mostrar Propiedades de *Widget*	EC 1.1 Mostrar Propiedades de un *Widget*	En cualquier caso que el usuario tenga seleccionado solamente un *widget* debe mostrarse a la derecha el menú con las propiedades de dicho *widget*.
	EC 1.2 No mostrar propiedades de más de un *widget*.	En cualquier caso que el usuario tenga seleccionado más de un *widget* no debe mostrarse el menú de propiedades.

Tabla 10: Validación del CU mostrar propiedades de *widget*

Propiedades: Atributos que identifican a cada *widget* como entidad única.

Figura 41: Mostrar propiedades antes

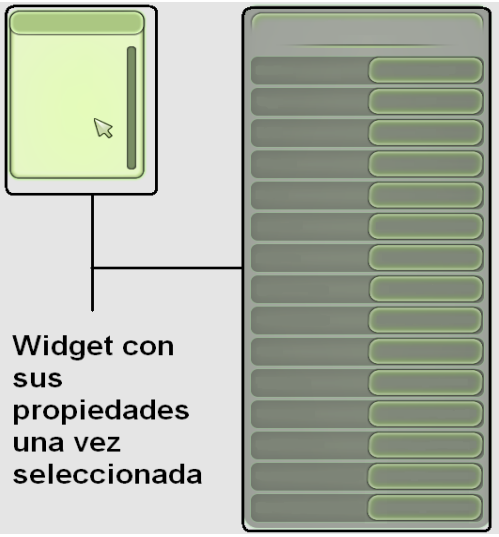

Figura 42: Mostrar propiedades después

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad
SC 1. Modificar Propiedades de *Widget*	EC 1.1 Modificar Propiedades de un *Widget*	Una vez que el usuario haya seleccionado un *widget* podrá editar sus propiedades tecleando valores válidos en este menú y tras presionar la tecla Enter los cambios serán aplicados sobre el *widget*.
	EC 1.2 Formato inválido	En cualquier caso que el usuario introduzca un valor con un formato no válido el sistema mostrará un mensaje de error.
	EC 1.3 Recurso no disponible	En caso de que el usuario introduzca como valor un

		recurso que no ha sido cargado, el sistema permitirá definir una ruta para cargar el recurso y luego este será aplicado.

Tabla 11: Validación del CU modificar propiedades de *widget*

Formatos:

- En el caso de los materiales los nombres serán los mismos que se usen en el archivo .material que los contenga ej.: "*SdkTrays/Tray*".
- En el caso de las fuentes los nombres serán los mismos que se usen en el archivo .fontdef que los contenga ej.: "*SdkTrays/Caption*".
- En el caso de las dimensiones y posiciones serán válidos solo números enteros ej.: "64", "500".
- En las listas cada elemento estará separado por una coma ej.: "*item1*,*item2*,*item3*".
- Los colores estarán en el formato RGBA donde cada componente será un valor real entre 0 y 1 separados por una coma ej.: "0.500,0.950,0.1,1".

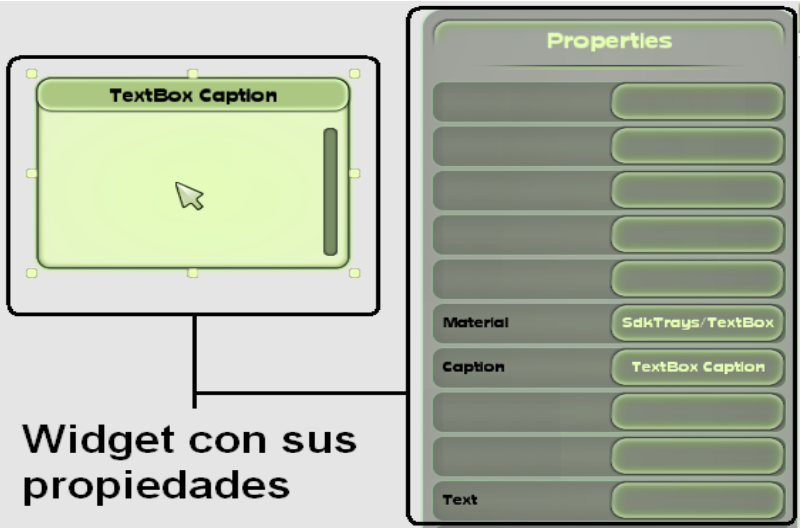

Figura 43: Modificar propiedades antes

Figura 44: Modificar propiedades después

Figura 45: Formato no válido

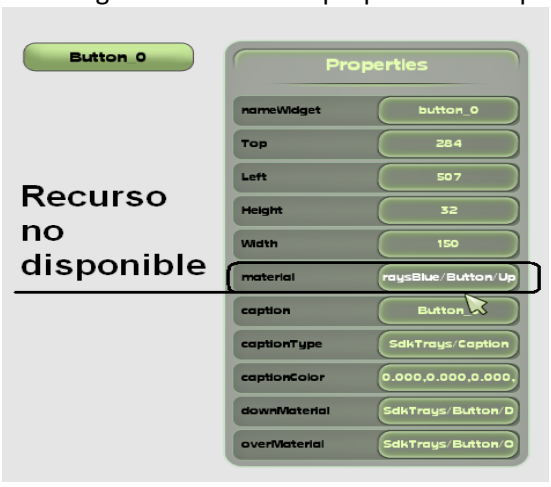

Figura 46: Recurso no disponible

Figura 47: Informando recurso no disponible

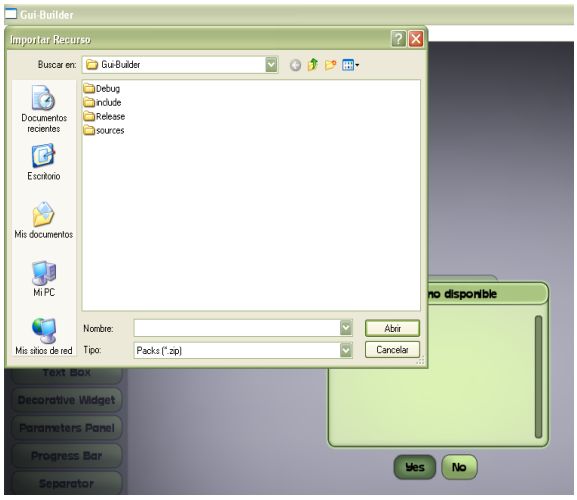

Figura 48: Cargando recurso

Figura 49: Recurso cargado y aplicado

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad
SC 1. Mostrar Vista Preliminar	EC 1.1 Mostrar Vista Preliminar	El usuario siguiendo la ruta "*View/Show preview*" podrá ver la configuración realizada tal y como se mostraría en la aplicación.
	EC 1.2 Finalizar Vista Preliminar	El usuario podrá volver al modo de edición siguiendo la ruta "*View/End preview*".
	EC 1.3 Interactuar con *Widget*	El usuario podrá interactuar con los *widgets* durante la vista preliminar pero solo con sus características visuales, estos no desencadenarán ninguna acción.

Tabla 12: Validación del CU mostrar vista preliminar

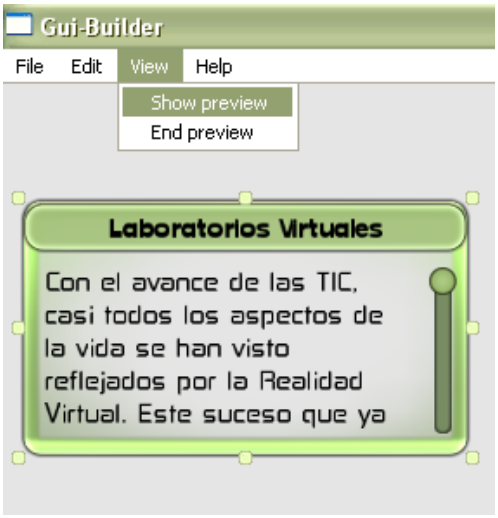

Figura 50: Mostrar vista preliminar

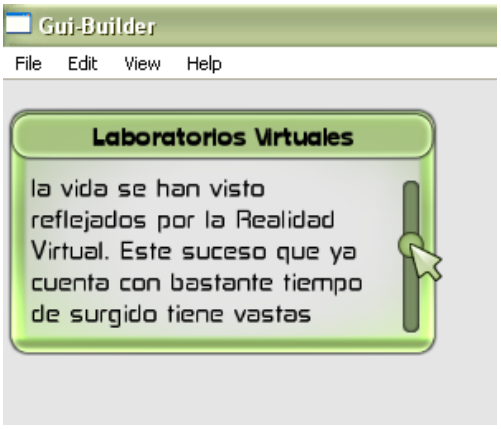

Figura 51: Interactuar con *widget*

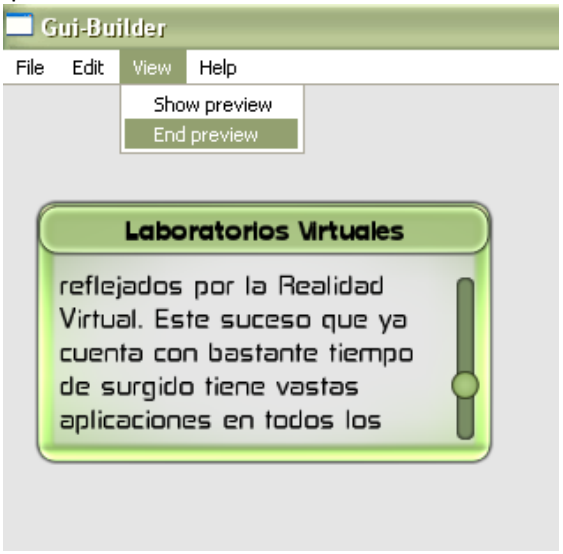

Figura 52: Finalizar vista preliminar

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad
SC 1. Exportar Configuración	EC 1.1 Exportar Configuración	El usuario siguiendo la ruta "*File/Save*" o presionado la combinación de teclas "Ctrl +

		S" podrá salvar una configuración previamente creada. Para esto debe especificar una ruta y un nombre de archivo en el diálogo que se le mostrará.
	EC 1.2 Cancelar Exportar	Una vez mostrado el diálogo si el usuario presiona el botón cancelar, la operación será suspendida.

Tabla 13: Validación CU exportar configuración

Dirección: Es el directorio donde se guardará el archivo de configuración ej.: "D:\User\Projects\".

Nombre: Es el nombre que se le pondrá al archivo de configuración ej.: "Configuración1.uic".

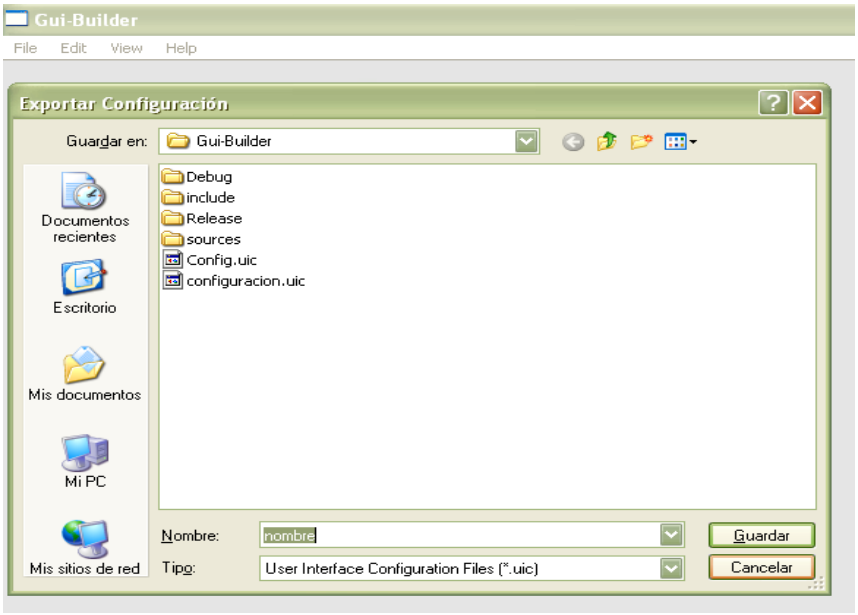

Figura 53: Exportar configuración

Nombre de la Sección	Escenarios de la Sección	Descripción de la Funcionalidad

SC 1. Importar Configuración	EC 1.1 Importar Configuración	El usuario siguiendo la ruta "File/Load" o presionado la combinación de teclas "Ctrl + O" podrá cargar una configuración desde un archivo válido. Para esto debe especificar una ruta y un nombre de archivo en el diálogo que se le mostrará.
	EC 1.2 Cancelar Importar	Una vez mostrado el diálogo si el usuario presiona el botón cancelar, la operación será suspendida.

Tabla 14: Validación del CU importar configuración

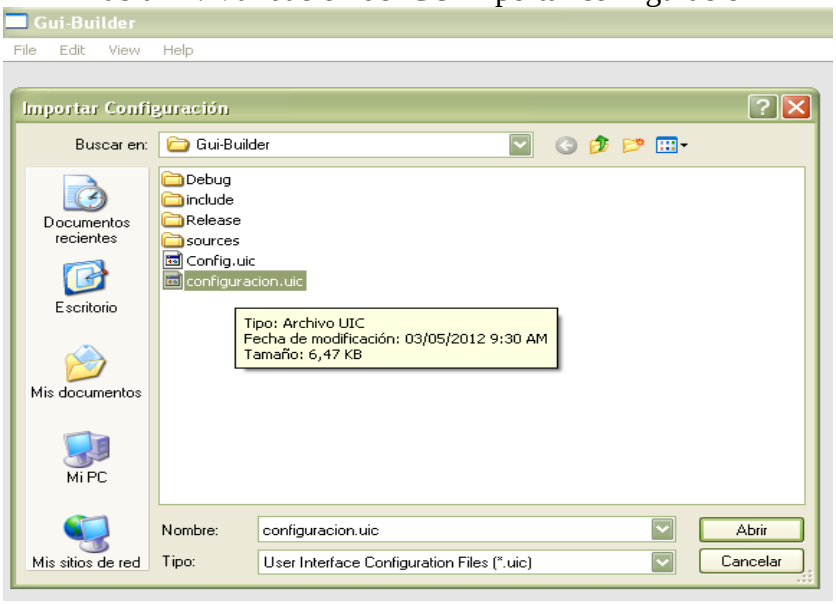

Figura 54: Importar configuración

Nombre de la Sección	Escenarios de la	Descripción de la

	Sección	Funcionalidad
SC 1. Cargar Configuración	EC 1.1 Cargar Configuración	El usuario utilizando una instancia de la clase UICLoader previamente agregada a su proyecto puede invocar el método loadPrebuildConfiguration() pasándole como parámetros la ruta del archivo de configuración.
	EC 1.2 Ruta no válida.	Una vez ejecutándose la aplicación si la ruta especificada no es correcta se lanzará una excepción informándolo.
	EC 1.3 Formato no válido	Una vez ejecutándose la aplicación si el formato del archivo cargado no es correcto se lanzará una excepción informándolo.

Tabla 15: Validación del CU cargar configuración

Conclusiones generales del capítulo.

Tras la implementación y pruebas del sistema han quedado satisfechos todos los requerimientos, para esta tarea ha sido de gran utilidad cada uno de los artefactos generados en capítulos anteriores. Ya que las pruebas arrojaron resultados satisfactorios se dieron por concluidas las iteraciones del proceso de desarrollo.

CONCLUSIONES

Con la culminación del presente trabajo se desarrolló un software que permite diseñar interfaces gráficas de usuario, para incluirlas en aplicaciones que utilicen el motor de *render* Ogre3D, específicamente para los laboratorios virtuales. El sistema obtenido solamente depende de Ogre3D y Qt por lo que reduce las dependencias de software de terceras partes. Por otra parte se determinó que los componentes gráficos construidos a partir de *overlays* de Ogre3D poseen una gran ventaja y es que su apariencia puede ajustarse a las necesidades de los diseñadores con cambios en pocas propiedades visuales.

RECOMENDACIONES

Permitir el cambio de estilo de las *widgets* de manera conjunta, y realizar animaciones a estas.

Además se propone integrarlo con el resto de los editores desarrollados en el proyecto, de manera que sean una única herramienta.

Por otra parte se recomienda que permita añadir componentes creados por el usuario, para adecuarlo más a sus necesidades.

BIBLIOGRAFÍA

1. **Real Academia Española.** Diccionario de la Real Academia Española - Vigésima segunda edición. [Online] 2001. [Cited: Febrero 27, 2012.] http://buscon.rae.es/draeI/.
2. **Rodeiro Iglesias, Javier.** Escola Superior en Enxeñería Informática - Ourense. [Online] Marzo 11, 2004. [Cited: Febrero 27, 2012.] http://trevinca.ei.uvigo.es/~jrodeiro/Teaching/diu/DIU-Tema1.pdf.
3. **Aimacaña Toledo, Carlos.** Interfaz de Usuario. *monografias.com.* [Online] 2006. [Cited: Febrero 26, 2012.] http://www.monografias.com/trabajos6/inus/inus.shtml..
4. **Johnson, Jeff, et al.** Digibarn Computer Museum. [Online] Septiembre 1989. [Cited: Febrero 27, 2012.] http://www.digibarn.com/friends/curbow/star/retrospect/.
5. **Marrero Expósito, Carlos.** *Interfaz Gráfica de Usuario: aproximación semio-cognitiva.* [Documento] Tenerife : s.n., 2006.
6. **Vromans, Johan.** Vromans.ORG. *GUI development with wxGlade.* [Online] Septiembre 28, 2006. http://www.vromans.org/johan/articles/wxglade.pdf.
7. **Alvarez, Alejandro B. and Valerio, Esteban R.** Generadores de Interfaces de Usuario: QT Designer, NetBeans y Windows Forms Designer. [Online] Junio 20, 2007. [Cited: Febrero 28, 2012.] http://www.di-mare.com/adolfo/cursos/2007-1/pp-GenGUI.pdf.
8. **Posadas, Sylvia.** Sibagraphics. *The Meaning of Colours.* [Online] Marzo 2, 2009. [Cited: Febrero 26, 2012.] http://www.sibagraphics.com/colour.php.
9. **Apple Inc.** Developer Tools. [Online] 2012. [Cited: Febrero 27, 2012.] https://developer.apple.com/technologies/tools/.
10. **The GNOME Project.** GNOME: The Free Software Desktop Project. *GTK+.* [Online] 2011. [Cited: Febrero 28, 2012.] http://developer.gnome.org/platform-overview/stable/gtk.
11. **The Glade project.** Glade - A User Interface Designer. [Online] 2009. [Cited: Febrero 27, 2012.] http://glade.gnome.org/.
12. **Nokia Corporation and/or its subsidiaries.** Qt Creator IDE and tools . [Online] Septiembre 2010, 2010. [Cited: Marzo 1, 2012.] http://qt.nokia.com/products/developer-tools.
13. **Trolltech.** [Online] 2005. [Cited: Marzo 1, 2012.] http://doc.trolltech.com/3.3/designer-manual-1.html.
14. **Ollivier, Kevin.** wxEmbedded: wxWidgets for embedded applications. [Online] [Cited: Marzo 1, 2012.] http://www.wxwidgets.org/docs/embedded.htm.
15. **Spitzak, Bill.** Fast Light Toolkit. [Online] 2011. [Cited: Marzo 1, 2012.] http://www.fltk.org/.
16. **Crazy Eddie's Gui System for Games.** Features. [Online] Marzo 3, 2011. [Cited: Marzo 1, 2012.] http://www.cegui.org.uk/wiki/index.php/Features.
17. **Streeting, Steve.** OGRE Manual v1.7 ('Cthugha'). [Online] Octubre 15, 2010. [Cited: Enero 20, 2012.] http://www.ogre3d.org/docs/manual/.
18. **Moen, Jacob.** Support and community documentation for Ogre3D. [Online] 2010. [Cited: Febrero 26, 2012.] http://www.ogre3d.org/tikiwiki/SdkTrays.
19. **Soto, Lauro.** Tecnologico. [Online] 2010. [Cited: Febrero 27, 2012.] http://www.mitecnologico.com/Main/DefinicionDeInterfaz. 1.
20. **Fadeyev, Dmitry.** 8 Characteristics Of Successful User Interfaces. *UsabilityPost.* [Online] 2008. [Cited: Febrero 27, 2012.] http://www.usabilitypost.com/8 Characteristics Of Successful User Interfaces.htm.
21. **Nokia Corporation and/or its subsidiaries.** Autodesk. [Online] 2011. [Cited: Febrero 29, 2012.] http://qt.nokia.com/qt-in-use/autodesk/.
22. —. VLC Media Player. [Online] 2011. [Cited: Febrero 29, 2012.] http://qt.nokia.com/qt-in-use/story/app/vlc-player/.

23. **The Code::Blocks team.** The open source, cross platform, free C++ IDE. [Online] 2011. [Cited: Marzo 1, 2012.] http://www.codeblocks.org/.
24. **Microsoft.** Modeling the Application. [Online] 2012. [Cited: Marzo 1, 2012.] http://msdn.microsoft.com/en-us/library/57b85fsc.aspx.
25. **Vary, James P.** *Informe de la reunión de expertos sobre laboratorios virtuales.* Paris : s.n., 2000.
26. **Scheckler, Rebecca K.** *Virtual labs: a substitute for traditional labs?* [Documento] Virginia : UBC Press, 2003.
27. **Crystal Space Team.** Crystal Space 3d. [Online] 2011. [Cited: Marzo 1, 2012.] http://www.crystalspace3d.org/.
28. **Delta3D Development Team.** Delta 3D-Open source gaming & simulation engine. [Online] 2010. [Cited: Marzo 1, 2012.] http://www.delta3d.org/.
29. **Torus Knot Software Ltd.** OGRE-Open Souce 3D Graphics Engine. [Online] TYPO3 company, 2009. [Cited: Febrero 2, 2012.] http://www.ogre3d.org/.
30. **Garage Games.** GarageGames.com. [Online] 2012. [Cited: Marzo 2, 2012.] http://www.garagegames.com/products/torque-3d.
31. **Epic Games, Inc.** Game Engine Technology by Unreal. [Online] 2012. [Cited: Marzo 2, 2012.] http://www.unrealengine.com/.
32. **CRYTEK.** CryTek. [Online] 2012. [Cited: Marzo 2, 2012.] http://www.crytek.com/.

GLOSARIO

3D: En computación gráfica las tres dimensiones son el largo, el ancho y la profundidad de una imagen. Técnicamente hablando el único mundo en 3D es el real, la computadora sólo simula gráficos en 3D, pues, en definitiva toda imagen de computadora sólo tiene dos dimensiones, alto y ancho que no es más que la resolución.

TIC: Tecnologías de la Información y de la Comunicación conforman el conjunto de recursos necesarios para manipular la información, particularmente los ordenadores, programas informáticos y redes necesarias para convertirla, almacenarla, administrarla, transmitirla y encontrarla.

Widget: Abreviación de las palabras *window* y *gadget* (ventana y dispositivo, en inglés). En efecto, un *widget* es una mini-aplicación de ordenador que se presenta como una pequeña ventana o caja.

Framework: En el desarrollo de software, un *framework* es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.