UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 ENTORNOS VIRTUALES

Módulo para la creación y control de interfaces visuales para Sistemas de Realidad Virtual

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

Autor: Yenifer del Valle Guevara

Tutor: Ing. Yanoski Rogelio Camacho Román

Ciudad de la Habana

Mayo de 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

Yenifer del Valle Guevara

_____

Firma del Autor

Yanoski Rogelio Camacho Román

_____

Firma del Tutor

DATOS DE CONTACTO

Nombre y Apellidos: Yanoski Rogelio Camacho Román
Edad: 26 años
Ciudadanía: cubano
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática
Categoría Docente: Profesor Instructor
E-mail: rcamacho@uci.cu

Graduado de la CUJAE, con tres años de experiencia en el tema de la Gráfica Computacional, y líder de un proyecto de Realidad Virtual en la Universidad de las Ciencias Informáticas.

Agradecimientos

A Alexey por su apoyo en todo momento. A todas las personas que me han ayudado en la realización de este trabajo.

Dedicatoria

A Tania. A mi mamá. A mi familia.

Resumen

La necesidad de minimizar los costos de realización de los productos informáticos impone a los programadores el reto de realizar módulos para la reducción de tiempo.

El presente trabajo presenta un Módulo para la creación y control de interfaces visuales para Sistemas de Realidad Virtual, el cual será insertado en la herramienta *SceneToolkit* que se esta desarrollando en el proyecto Herramientas de desarrollo, y será muy útil ya que reducirá considerablemente los costos de realización de dichos sistemas al agilizar la creación de interfaces gráficas.

El módulo facilitará al usuario la creación de componentes visuales como botones, menús, etiquetas y paneles. También brindará opciones como la visualización de imágenes, transformaciones geométricas, entre otros.

Para lograr el buen funcionamiento del módulo, se estudian las bibliotecas de clases existentes y se realiza la ingeniería de software de dicho módulo.

Se implementa una versión primaria del módulo programando sólo aquellos componentes visuales más utilizados en estos tipos de interfaces visuales, que son los botones, menús, paneles y etiquetas.

Palabras clave

Módulo, interfaz visual, componentes, panel, menú, botón, etiqueta.

Tabla de contenido

Introducción

En nuestro país se ha ido introduciendo paulatinamente el concepto de Realidad Virtual (RV) y se ha trabajado para fomentar su desarrollo, debido a las grandes aplicaciones que de ella se derivan en varios sectores como la salud, la industria armamentista, la educación, la meteorología, los juegos, etc.

Un ejemplo de esto es la empresa SIMPRO (Simuladores Profesionales) que se dedica a la producción de simuladores basados en técnicas de RV y que cuenta con una basta experiencia en el tema.

La Universidad de las Ciencias Informáticas (UCI), institución educacional dedicada además a la producción de software, tiene entre sus perfiles el desarrollo de Sistemas de Realidad Virtual (SRV) y para ello ha entrado en colaboración con la empresa SIMPRO, creándose así varios proyectos investigativos y productivos conformados por especialistas de dicha empresa y estudiantes de diferentes años de la Universidad.

Para viabilizar el trabajo de estos proyectos, haciéndolo más efectivo en cuanto a tiempo de realización, existe un proyecto (Herramientas de Desarrollo para Sistemas de Realidad Virtual) que ha venido desarrollando una biblioteca o herramienta gráfica para entornos de RV y que agrupa un conjunto de funcionalidades comunes a cualquier juego o simulador para uso de los demás proyectos en el desarrollo de productos finales.

El simulador de conducción de auto, desarrollado por uno de estos proyectos en respuesta a demandas de escuelas de conducción y de clientes en el extranjero, consta, a efectos de este tema de tesis, de dos partes importantes: un entorno virtual donde la persona a evaluar realiza un ejercicio práctico que consiste en manejar un auto conectado al SRV; y una interfaz para un instructor o evaluador que le permite a éste configurar el ejercicio (seleccionar nivel de complejidad) y monitorear su desarrollo (se le muestran controles del auto como los relojes de velocidad y revoluciones por minutos (rpm), los intermitentes izquierdo y derecho, velocidad, si el carro está encendido o no, estado de la emergencia, el cloche, el acelerador y puede controlar el ejercicio, por ejemplo, deteniendo la actividad).

Esta funcionalidad del software, conocida como “puesto del instructor”, y que se ejecuta de forma paralela a la simulación, forma parte actualmente del simulador de auto, pero se pretende que otros simuladores e incluso los juegos puedan tener la opción de brindar este puesto, ya sea para que alguien monitoree el trabajo de otro, o para que el propio usuario configure su ejercicio o juego. Por tanto, es conveniente contar con un módulo que automatice la creación de estas interfaces, permitiendo añadir elementos (menú, botones) que serán usados por los clientes para controlar dichos productos.

El proyecto de Herramientas de Desarrollo para Sistemas de Realidad Virtual antes mencionado, no cuenta con un módulo de este tipo, por tanto se tiene la necesidad de implementar un Módulo para la creación y control de interfaces gráficas, cuyo resultado, la interfaz generada, pueda ser acoplada a simuladores o juegos de manera fácil para los programadores de los productos finales.

El problema científico a resolver es cómo implementar un Módulo para la creación y control de interfaces visuales para SRV.

Constituye objeto de estudio de este tema las herramientas visuales para la incorporación de elementos de control como botones, etiquetas, paneles y menú a la realización de aplicaciones, siendo el campo de acción el desarrollo de un módulo para la creación y control de interfaces gráficas para la automatización del trabajo de los desarrolladores de sistemas de realidad virtual.

El objetivo de este trabajo es implementar un módulo de interfaces visuales para la configuración y monitoreo de SRV.

Para el total cumplimiento del mismo se trazan las siguientes tareas de investigación:

- Analizar herencia de componentes visuales en los entornos de desarrollo más conocidos, así como principales atributos y métodos de estos.
- Estudiar la estructura de la biblioteca desarrollada por el proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual, como punto de partida para el diseño de la arquitectura del nuevo módulo.
- Realizar la ingeniería de software requerida para el desarrollo del nuevo módulo.

- Implementar una versión primaria del Módulo para la creación y control de interfaces visuales para SRV.

Con el desarrollo de este módulo se espera que los programadores no tengan que emplear mucho tiempo en la realización de interfaces visuales, sino que con bastante facilidad puedan desarrollar componentes visuales de manera que no tengan que preocuparse por temas como la visualización de imágenes, transparencia, transformaciones geométricas, conversión de escalas para el acople con un sistema de realidad virtual y simplemente utilizando las potencialidades del módulo, lograr un buen diseño de interfaz.

Capítulo 1 Fundamentación Teórica

Introducción

En *Windows*, no son las aplicaciones quienes manejan los recursos del sistema como pueden ser la memoria, los ficheros, las ventanas y hasta los controles gráficos. Es el sistema operativo quién realiza todas estas tareas. Y la aplicación se limita a realizar peticiones de lo que necesita. Estas peticiones se realizan mediante lo que se denomina la *API de Windows*, que es un conjunto de funciones que proporciona el sistema operativo para poder habilitar la comunicación. Pero este conjunto de funciones es muy grande, poco estructurado y difícil de manejar. [1]

Por todo esto es que los entornos de programación visuales, en su afán de aportar simplicidad y estructuración a la programación en *Windows*, han desarrollado diversos **Marcos de Trabajo**. Un marco de trabajo es una fase intermedia que se coloca por encima de la API para aportar mayor sencillez a las aplicaciones. Normalmente estos marcos de trabajo son orientados a objetos y se implementan como bibliotecas del lenguaje base, por ello también se suelen denominar **bibliotecas de clases**. [1]

Dado que el módulo para la creación y control de interfaces visuales que ocupa a este proyecto, se compone principalmente de una biblioteca de clases para el manejo de controles visuales, en el presente capítulo se hace una investigación acerca de cómo se han estructurado las bibliotecas de clases en marcos de trabajos tales como el compilador de C++ de la *Borland* y el *Visual C++* de *Microsoft* por ser de las más conocidas en la actualidad.

En este capítulo se exponen los resultados de todo lo anteriormente expuesto y se llegan a conclusiones.

1.1 Biblioteca de Componentes Visuales de la Borland

En 1995, Borland lanzó al mercado *Delphi*, que supuso la revolución en la programación para *Windows* e inició el desarrollo rápido y sencillo de aplicaciones visuales. *Delphi* ofrecía el desarrollo rápido de aplicaciones empleando componentes (objetos que pueden ubicarse en formularios y manipulados por medio de propiedades, métodos y eventos). Esta forma de trabajo se hizo popular con *Visual Basic*. *Delphi* empleaba como base al lenguaje *Object Pascal* (una ampliación de *Pascal* que incorpora programación orientada a objetos) y permitía la creación de programas ejecutables independientes que no requerían intérprete, por lo que se ejecutaban mucho más rápido que los de *Visual Basic*. [1]

Lo más relevante desde el punto de vista técnico es que *Borland* creó la biblioteca VCL que es un marco de trabajo para crear aplicaciones *Windows* diseñadas en torno al concepto de componente (propiedades, métodos y eventos). La biblioteca VCL desarrollada para *Delphi* es la misma que se emplea como núcleo de C++ *Builder*, de hecho, está escrita en *Object Pascal*. [1]

La biblioteca VCL hace un uso extensivo del concepto de herencia. El objetivo final de la biblioteca VCL es crear clases que representan a componentes; aunque algunas clases no hagan referencia a componentes concretos, realizan tareas de gestión interna y se emplean como clases bases para derivar mediante herencia otras clases. En la figura 1 se muestra una pequeña parte de la jerarquía de clases que conforma la biblioteca VCL de la *Borland*. Luego se brinda una breve explicación de las características de cada clase. [1]

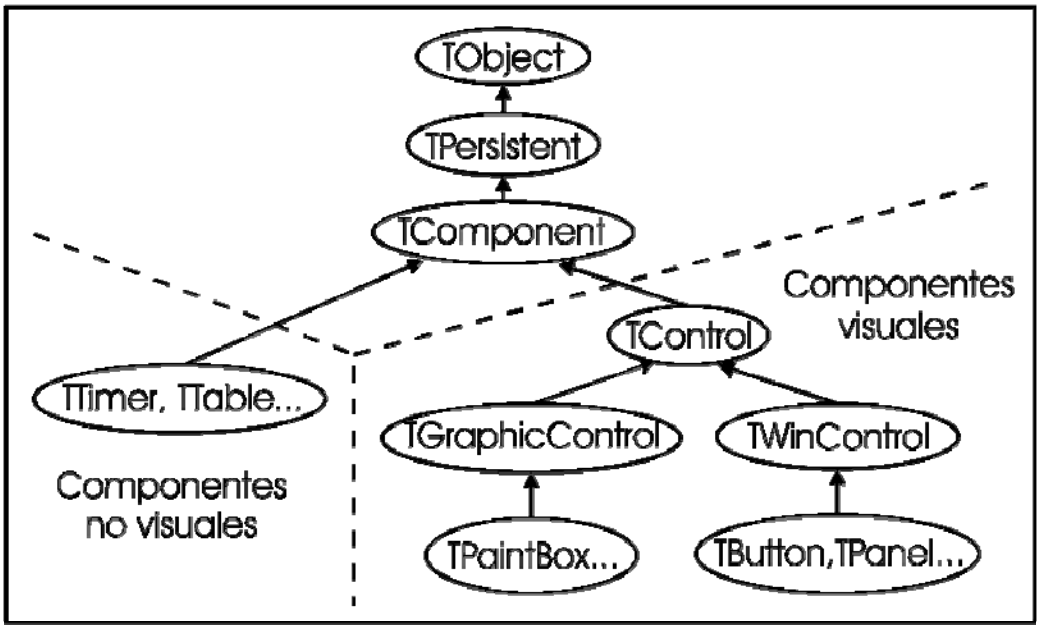

Figura 1 Jerarquía de clases de la VCL de la *Borland*

Las clases que conforman la parte superior de la jerarquía de la biblioteca VCL (*TObject*, *TPersistent* y *TComponent*) se denominan clases "abstractas" porque sirven para estructurar y agrupar comportamientos comunes de las clases de la biblioteca VCL. No se suelen crear objetos directamente a partir de ellas.

Rigurosamente hablando, no son clases abstractas porque no tienen métodos virtuales puros, pero se les puede considerar como tales en la práctica.

TObject

Es el ancestro de todas las clases de la biblioteca VCL. Encapsula el comportamiento común de los objetos en *C++ Builder*, como puede ser la información de la clase y la creación de instancias.

Suele ser una buena idea derivar las clases de *TObject*, porque aunque no sea normal que el programador haga uso de los métodos proporcionados por *TObject*, si lo es que lo haga *C++ Builder* en tiempo de ejecución.

Directamente de *TObject* heredan aquellas clases que no son componentes y no necesitan ser almacenadas en disco.

TPersistent

Esta clase tiene que ver con la habilidad de un objeto de almacenarse en disco o en memoria, asignarse a otros objetos así como otros detalles internos de *C++ Builder* que no es preciso conocer.

TComponent

Esta es una de las clases más importantes de la biblioteca *VCL* ya que la mayoría de los objetos que se manejan en una aplicación son componentes. Esta clase proporciona toda la funcionalidad que requiere un componente básico.

La funcionalidad de *TComponent* permite que los objetos aparezcan en la paleta de componentes y que sean manipulados por el diseñador de formularios, además de otras capacidades comunes a los componentes.

Los componentes no visuales se derivan directamente de *TComponent* mientras que los componentes visuales se derivan de *TControl*, que a su vez se deriva de *TComponent*. Por esta razón se suelen denominar *controles* a los componentes visuales.

TControl

Proporciona la funcionalidad de los componentes visuales (controles). Esta funcionalidad es principalmente el aspecto visual que deben ofrecer los componentes en tiempo de ejecución. *TControl* proporciona mayor funcionalidad que la que requieren los componentes visuales: los componentes individuales se derivan de *TGraphicControl* o de *TWinControl*, clases derivadas de *TControl*.

TGraphicControl

Generaliza a los controles que tienen una representación visual, pero no pueden recibir el foco: el usuario podrá verlos pero no interactuar con ellos. Suelen ser controles para visualizar texto (no editable en

tiempo de ejecución), gráficos o dibujos. Poseen una propiedad *Canvas* que permite acceder a su área de dibujo.

TWinControl

Son los controles típicos de *Windows* que pueden recibir el foco, contener otros controles y además poseen un manejador de ventana. Un manejador de ventana es un identificador que proporciona *Windows* para el control, por lo que podemos decir que *Windows* tiene un conocimiento directo de la existencia del mismo.

TApplication

La clase *TApplication* encapsula una aplicación Windows por lo que caracteriza las operaciones fundamentales de un programa en Windows. *TApplication* simplifica la interfaz entre el programador y el sistema operativo, ocupándose de tareas como gestionar el paso de mensajes, proporcionar la ayuda contextual, establecer los textos de sugerencia de los botones y barras de estado, procesamiento de "teclas rápidas", gestión de excepciones, ejecutar cuadros de mensajes, etc.

TForm

La clase *TForm* caracteriza a los formularios en la biblioteca VCL. Los formularios se emplean como ventanas principales, cuadros de diálogo, ventanas secundarias o cualquier otro tipo de ventana que se pueda imaginar.

1.1.1 Los componentes en la biblioteca VCL

Los componentes son elementos genéricos con una funcionalidad muy concreta, cuya única finalidad es la reutilización. Cada uno de ellos está destinado a realizar una tarea típica en una aplicación.

Un componente de la biblioteca VCL es una *clase* que caracteriza a un control de *Windows* agregando propiedades, métodos y gestores de eventos a cada control.

Los componentes se agrupan en la paleta de componentes en distintas páginas: [1]

- ***Standard***: Incluye los componentes comunes y más habituales de los programas *Windows*.
- ***Additional***: Los componentes de esta página son controles especializados propios de *C++ Builder*.
- ***Win32***: Componentes de cuadros de diálogo propios de *Windows 95/NT*.
- ***System***: Esta página incluye controles muy especializados para interacción con el sistema.
- ***Internet***: Componentes para distintos protocolos de acceso a Internet.
- ***Data Access*** y ***Data Controls***: Componentes especializados para acceso a bases de datos.
- ***Midas***: Esta página incluye componentes que permiten el desarrollo de aplicaciones multicapas con *MIDAS*.
- ***Decisión Cube***: Incluye componentes para realizar análisis multidimensionales de datos para tomar decisiones.
- ***QReport***: Componentes para diseñar rápidamente informes y resúmenes.
- ***Dialogs***: Contiene cuadros de diálogo comunes en aplicaciones *Windows* listos para usar.
- ***Win 3.1***: Componentes propios de *Windows 3.1*, la mayoría de ellos se mantienen en *C++ Builder* sólo por compatibilidad con versiones anteriores.
- ***Samples***: Componentes de demostración (cómo hacer componentes personalizados).
- ***Active X***: Los componentes de esta página son objetos *Active X*, creados por otros desarrolladores puede que utilizando otros lenguajes de programación.

Se pueden establecer muchas clasificaciones para los componentes. Una de ellas es la de visuales o controles, frente a no visuales. [1]

Un componente es visual cuando tiene una representación gráfica en tiempo de diseño y ejecución (botones, barras de scroll, cuadros de edición) y se dice no visual en caso contrario (temporizadores, cuadros de diálogo no visibles en la fase de diseño). Por lo demás no existen más diferencias entre ellos, excepto, claro está, las derivadas de la visualización del componente.

Los componentes no visuales se pueden colocar en los formularios de la misma manera que los controles, aunque en este caso su posición es irrelevante.

De un componente se puede destacar tres aspectos: sus propiedades, los métodos que puede ejecutar y los eventos a los que puede responder. [1]

1.1.1.1 Propiedades

Las propiedades son los elementos del componente que configuran su aspecto y controlan su comportamiento.

Muchos componentes tienen propiedades en común. Por ejemplo, todos los componentes visuales tienen las propiedades *Top* y *Left* que controlan la posición del componente en el formulario, tanto en tiempo de diseño como en tiempo de ejecución.

Se ha visto como pueden establecerse y modificarse las propiedades de los componentes en tiempo de diseño utilizando el inspector de objetos. Las propiedades tienen, normalmente, un método de acceso asociado que se ejecuta al modificar la propiedad en tiempo de ejecución. Para modificar una propiedad basta (generalmente) con asignarle el nuevo valor. Al realizar un cambio, la biblioteca VCL invoca el método de acceso a la propiedad. En el caso concreto de la propiedad *Left*, la biblioteca VCL dibuja de nuevo todo el formulario en la nueva ubicación.

A modo de resumen, las propiedades tienen dos especificadores de acceso que se emplean al consultar o modificar el valor de una propiedad en tiempo de ejecución. En definitiva, hay un especificador de lectura y un especificador de escritura.

Los especificadores de acceso asocian métodos de lectura o de escritura (funciones, en definitiva) con las propiedades. Al consultar o modificar el valor de una propiedad se invocan, automáticamente, las funciones respectivas asociadas:

- Al asignar un nuevo valor a una propiedad se está accediendo al especificador de escritura. La biblioteca VCL comprueba si existe un método de acceso asociado al especificador de escritura y si es así, lo ejecuta. Si no existe, simplemente asigna el nuevo valor a la propiedad.
- Al consultar el valor de una propiedad se está accediendo al especificador de lectura. En la mayoría de los casos el especificador de lectura no hace más que devolver el valor de la propiedad.

Aunque cada componente tiene sus propias propiedades, métodos y eventos existen algunas propiedades comunes:

Propiedad	**Descripción**
Align	Establece distintas posiciones y ajustes al componente.
Caption	Indica el texto que acompaña a algunos componentes.
Name	Indica el nombre o identificador que se le asigna al componente en el programa y con el que se referirá al componente en la aplicación (en el código).
Color	Se refiere al color de fondo del componente.
Cursor	Establece que icono se muestra cuando el usuario mueve el cursor sobre un componente.
Font	Permite especificar las propiedades de la fuente de letra que se usará en el componente.
Enabled	Para activar y desactivar componentes: cuando un componente se desactiva

	no puede recibir el foco, por lo que no responde al *mouse*, al teclado ni a eventos del temporizador.
Hint, ***ShowHint***	***Hint*** especifica el texto que debe aparecer en el cuadro de ayuda o texto de sugerencia asociado a un componente. Tiene dos partes, separadas por la barra |. La primera (sugerencia breve) se visualiza cuando el usuario coloca el puntero sobre el componente. La segunda se presenta en la barra de estado. Esta información aparece cuando ocurre el evento ***OnHint***. ***ShowHint*** determina si la ayuda de sugerencia breve se debe mostrar o no.
ParentColor, ***ParentFont,*** ***ParentShowHint***	Cuando se ponen en verdadero, heredan el valor para la propiedad a la que hacen referencia de su componente padre. En otro caso, emplean el valor que tienen establecido para esa propiedad.
Visible	Indica si el componente es visible o no.

Tabla 1 Propiedades de los componentes de la VCL de la Borland.

Las siguientes propiedades, aunque se usan frecuentemente, no están disponibles para todos los componentes de la biblioteca VCL.

Propiedad	**Descripción**
BorderStyle	Para diferenciar al componente del formulario o integrarlo en su fondo.
Height y ***Width***	Altura y anchura (en píxeles) del componente.
HelpContext	Para asociar un número de índice de un fichero de ayuda a un componente.
Left y***Top***	Coordenadas *X* e *Y* del componente (su esquina superior izquierda).

TabOrder, ***TabStop***	Para establecer el orden de tabulación y determinar si el componente forma o no parte de la secuencia de tabulación, respectivamente.

Tabla 2 Otras propiedades de los componentes de la VCL de la Borland.

1.1.1.2 Métodos

Los métodos son funciones asociadas al componente que pueden invocarse para que el componente realice distintas acciones.

Por ejemplo, todos los componentes visuales tienen un método llamado *Show()* para mostrarlos y otro llamado *Hide()* para ocultarlos. En C++ los métodos son miembros de una clase, al igual que las propiedades.

La siguiente tabla muestra algunos métodos comunes de los componentes.

Método	Descripción
ClientToScreen(), ***ScreenToClient()***	Convierte las coordenadas del área de cliente en coordenadas de pantalla y viceversa.
Hide(), ***Show()***	***Hide()*** oculta el componente. Puede volver a hacerse visible posteriormente con ***Show()***. Ambos métodos modifican la propiedad ***Visible***.
Invalidate(), ***Repaint(),*** ***Update()***	Re-dibujar, actualizar o invalidar un componente.
SetBounds()	Establece simultáneamente los valores de las propiedades ***Top***, ***Left***, ***Width*** y ***Height***.

SetFocus()	Sitúa el foco sobre un componente y lo convierte en el componente activo.
CanFocus()	Devuelve verdadero si el componente puede recibir el foco (si las propiedades ***Visible*** y ***Enabled*** están en verdadero).

Tabla 3 Métodos de los componentes de la VCL de la Borland.

1.1.1.3 Eventos

Un **evento** es cualquier suceso que puede ocurrirle a un componente (movimiento del *mouse*, pulsación de algún botón del *mouse*, pulsación de una tecla, desplazamiento o redimensionamiento de una ventana), que pueden condicionar el comportamiento y apariencia del programa.

Cada componente poseerá una serie de eventos que puede recibir o generar. Se pueden tratar los eventos de un componente y dejar que los demás sean tratados por defecto.

Cuando se responde a un evento se dice que se está manipulando el evento. Los eventos se manejan mediante los gestores o manipuladores de eventos.

Se dice que *Windows* está orientado a eventos. Esto significa que cualquier programa está condicionado por los eventos que ocurren en el entorno *Windows*. Un programa *Windows* está continuamente sondeando al sistema ante la ocurrencia de cualquier evento, y de ocurrir, *Windows* avisa al programa enviándole un mensaje. Un programa *Windows* está ocioso la mayor parte del tiempo esperando a que ocurra algún evento.

En la siguiente tabla se muestran los eventos más importantes y comunes:

Evento	Descripción
OnMouseDown, ***OnMouseMove***, ***OnMouseUp***, ***OnClick,*** ***OnDblClick***	Para responder a los eventos que suceden al mover el *mouse* o cuando se pincha con él sobre el componente.
OnEnter, ***OnExit***	***OnEnter*** ocurre cuando se activa un componente (recibe el foco) siempre que el foco se transfiera desde otro componente del mismo formulario. ***OnExit*** ocurre cuando el componente pierde el foco, siempre que el foco se transfiera a otro componente del mismo formulario.
OnKeyDown, ***OnKeyUp,*** ***OnKeyPress***	Para responder a los eventos que suceden al pulsar alguna tecla cuando el foco está en un componente.
OnPaint	Este evento ocurre cuando un objeto tiene que ser redibujado. Si el objeto es un *PaintBox*, este gestor se encargará de dibujar sobre el *Canvas* y si no se proporciona este gestor, el *PaintBox* no se verá en tiempo de ejecución.

Tabla 4 Eventos de los componentes de la VCL de la Borland

1.2 Biblioteca Fundamental de Clases de *Microsoft* (MFC)

La *Microsoft Foundation Class Library* (MFC) es el marco de trabajo desarrollado por *Microsoft* e incorporado a los compiladores *Microsoft Visual C++* (actualmente incluso en *Borland C++ 5*). [1]

La MFC es una colección de clases que pueden ser usadas en la realización de programas de aplicación. Las clases en la biblioteca MFC están escritas en el lenguaje de programación C++. La biblioteca MFC le ahorra tiempo al programador pues le provee código que ya ha sido escrito. También provee un marco de trabajo para el desarrollo de programas de aplicación. [2]

En la biblioteca MFC también hay clases para todos los elementos de interfaz del usuario (ventanas, marcos, menús, barras de herramientas, barras de estado) para la construcción de interfaces para bases de datos, para el manejo de eventos como por ejemplo los mensajes de otras aplicaciones, para el manejo de las entradas por teclado y *mouse* y para la creación de controles *Active X*.

Esta biblioteca está compuesta por cientos de clases distintas, que desde el punto de vista funcional, se pueden dividir en 4 grupos: [3]

1. Clases orientadas al interfaz de usuario, representan ventanas, menús, diálogos, etc.
2. Clases de propósito general, representando ficheros, *strings*, datos de fecha y hora, etc.
3. Clases orientadas a bases de datos, representando tanto bases de datos como conjuntos de registros seleccionados después de una consulta sobre una tabla, etc.
4. Clases para manejo de excepciones, para control avanzado de errores de ejecución.

1.2.1 Filosofía de trabajo

El hecho de utilizar una librería de clases como la MFC añade complejidad al desarrollo e impone una serie de normas de programación a las que regirse. La complejidad añadida deriva de la necesidad de que el programador ahora no sólo debe controlar C/C++, sino que además debe conocer las clases de la MFC para poder utilizar su potencia.

Entran aquí en juego algunos conceptos conocidos: [3]

- **Herencia:** Un buen número de las clases de la MFC no se instancian, es decir, no se pueden crear objetos de esa clase con lo que no se podrá utilizar directamente. Esto significa que el programador deberá en muchos casos derivar sus propias clases de alguna de la MFC (estas clases "prohibidas" suelen ser abstracciones que dan pie a polimorfismos). Además se debe saber

que clase es la óptima para que sirva de clase base, por ejemplo, la clase MFC *CWinApp* es la que da soporte a las aplicaciones *Windows* y toda aplicación debe derivar su propia versión de la clase para poder funcionar (en realidad un programa MFC comienza con la creación de un objeto de ese tipo y finaliza con la destrucción del mismo). También se debe saber que si lo que se quiere es obtener una ventana de diálogo, se debe derivar la nueva clase desde *CDialog* (que es una clase especializada para ello) y no desde *CWnd* (que representa a una clase mas genérica y presentaría muchos problemas), por ejemplo, *CDialog* soporta que su plantilla haya sido dibujada en los recursos de la aplicación (al estilo de un formulario de Visual Basic) mientras que *CWnd* no, lo que obligaría a pintar los botones con código duro (programar su tamaño, posición en píxeles dentro de la ventana padre), aunque no sería imposible. En definitiva, a la hora de derivar clases de la MFC se debe tener en cuenta el concepto de especialización para obtener una programación más sencilla.

- **Polimorfismo:** La mayor potencia de la MFC reside en las funciones polimórficas (virtuales). El mecanismo de herencia no sería útil si no se pudiese redefinir el comportamiento por defecto de las clases de la MFC. Por ejemplo, si se crea un diálogo (derivando una clase desde *CDialog*) con un diseño propio, que tiene unos campos que se deben inicializar con los datos de un empleado para que aparezcan en pantalla, en este caso hay que definir un método de inicialización. Esto es posible, gracias a que este método (en concreto denominado OnInitDialog) es definido como virtual dentro de la clase base *CDialog*, lo que da lugar a una función polimórfica. Así, si en la clase derivada de *CDialog* se redefine ese método OnInitDialog, se asegura que el procedimiento de inicialización que se ejecutará será el que se redefinió en la clase derivada de *CDialog*. En definitiva, las dos clases (*CDialog* como clase base y la derivada) responden al mismo mensaje (OnInitDialog) pero de distinta manera (porque no comparten el método).

1.2.2 Clases básicas para interfaz de usuario

A continuación se brinda una breve explicación de las características de las clases básicas de la biblioteca MFC. [3]

CObject

Es la clase base principal de la biblioteca MFC. Hace el papel de raíz de no sólo de bibliotecas de clases como la CFile y la CObList, sino también de clases que puedan ser creadas por el programador. CObject suministra servicios básicos.

CCmdTarget

Es la clase base para la arquitectura de mapa de mensaje de la biblioteca MFC. Un mapa de mensajes encamina los comandos o mensajes de las funciones miembros que el programador escriba para ser manejados por ellos.

CWinApp

Representa una aplicación *Windows*. Cada programa debe tener un objeto global o estático (que exista durante todo el programa) de una clase propia derivada de *CWinApp*. De hecho, otra característica de la programación basada en MFC es que el código carece de programa principal (punto inicial de ejecución como la función "*main*" de C/C++ para DOS y UNIX) la ejecución comienza con la creación del objeto *CWinApp*. En realidad, es la MFC la que provee una función principal estándar *WinMain* (este es el nombre que toma para las aplicaciones *Windows*) que utilizarán todos los programas. Los pasos básicos que sigue son tres: inicialización del programa, entrada en el bucle de mensajes y finalización del programa. A continuación se muestran más detalles. [3]

- **Inicialización del programa:** Se obtiene en dos fases distintas: inicialización de la aplicación e inicialización de la instancia de la aplicación.
- **Bucle de mensajes:** Es la parte central de la ejecución, no finaliza hasta que no se cierra la ventana principal.

- **Finalización de la instancia:** Permite liberar recursos, cerrar ficheros de trabajo, confirmar operaciones.

CWnd

Es una de las clases importantes de la MFC ya que contiene las funcionalidades básicas de todas las clases de ventanas *Windows*. Como siempre, al ser una abstracción, no está muy orientada a la utilización directa, siempre se tiende a utilizar alguna de sus clases derivadas, especializadas, como *CFrameWindow* (ventana principal de aplicación) o *CDialog* (para ventanas de diálogo).

Es la clase responsable del funcionamiento del mecanismo de paso de mensajes de *Windows*. Este queda oculto por lo que se llama en la MFC mapa de mensajes. El mapa de mensajes de una clase de ventana recoge la asociación entre el mensaje recibido del sistema y la función que se ejecutará como respuesta. Todas estas funciones de respuestas (métodos de clase) son funciones virtuales que por tanto se pueden redefinir en todas las clases derivadas.

CFrameWnd

Se deriva de *CWnd*, proporciona las funcionalidades básicas de cualquier ventana principal de aplicación. Incluye soporte de menús, barra de estado y barra de herramienta.

Los controles

Se derivan de la clase *CWnd*. Son los componentes visuales los cuales presentan, al igual que en la biblioteca VCL, propiedades, métodos y eventos. Algunos de las clases que los manejan son: *CButton*, *CComboBox* y *CEdit*.

El Menú

CMenu se deriva directamente de *CObject*.

1.2.3 Jerarquía de Clases de la MFC

Este subepígrafe muestra como es tratada la herencia y el polimorfismo en la biblioteca de clases MFC. [4]

La siguiente figura muestra las distintas categorías en la jerarquía de clases de la MFC:

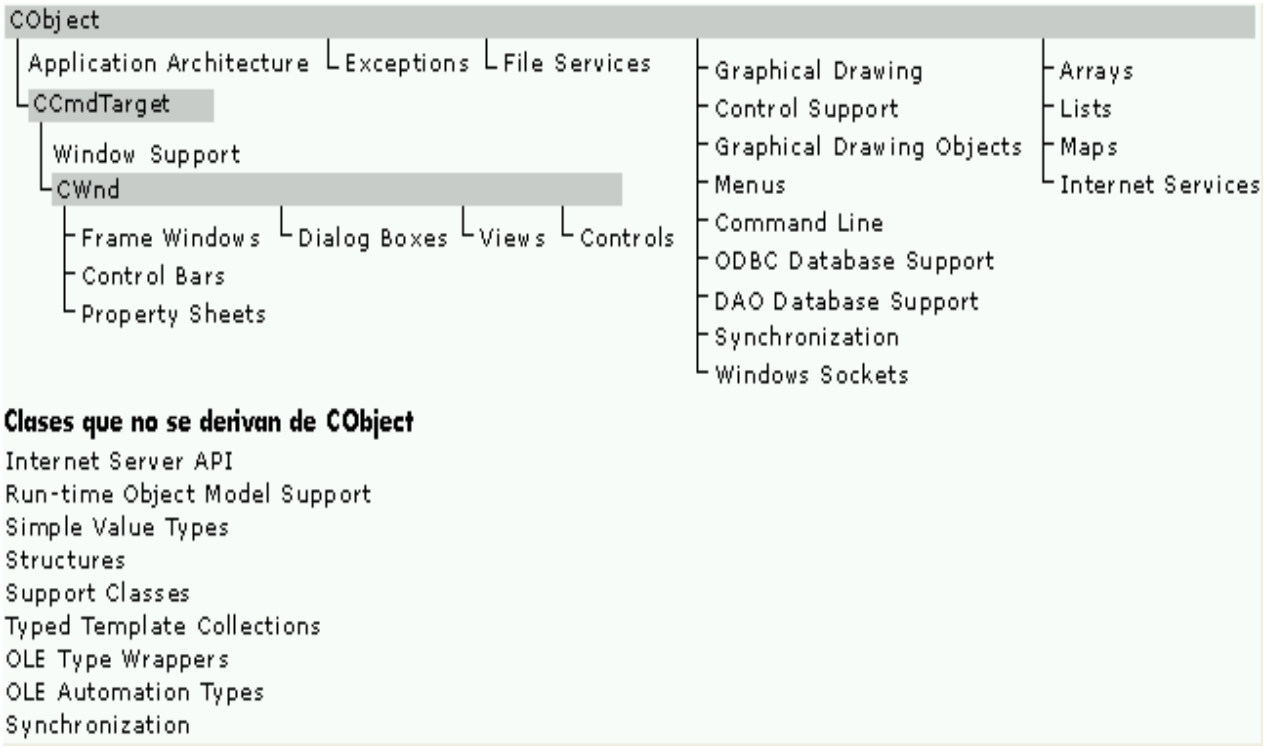

Figura 2 Categorías de la Jerarquía de Clases de la biblioteca MFC.

En la figura anterior se puede apreciar que la mayoría de los controles visuales se desprenden de la clase *CWnd*.

Conclusiones

En este capítulo se hizo una investigación acerca de como esta conformada la jerarquía de clases de las bibliotecas de dos de las herramientas más conocidas que tratan el tema de componentes en la actualidad: la biblioteca VCL de la *Borland* y la MFC de *Microsoft*. Para esto se tuvieron en cuenta aspectos tales como la herencia de clases y se hizo hincapié en el tratamiento que se les da a los componentes visuales en ambas bibliotecas de clases.

Se obtuvieron los elementos necesarios para dar respuesta al problema de este proyecto: el Módulo para la creación y control de interfaces visuales para SRV.

Capítulo 2 Características del sistema.

Introducción

En este capítulo se comienza con la explicación detallada del problema que trata este proyecto y se proponen soluciones técnicas para el sistema a elaborar.
Luego se concibe el producto a elaborar sobre la base de las dificultades y necesidades del cliente.

2.1 Objeto de estudio.

2.1.1 Problema y situación problémica.

Entre las demandas que recibe la UCI se encuentra la simuladores de conducción de autos, el cual consta de una parte práctica que se ejecuta en un sistema de realidad virtual y de otra parte conocida como “puesto del instructor”, que se ejecuta de forma paralela a la simulación.

Este “puesto del instructor” no es más que una interfaz gráfica en la cual el usuario puede interactuar con el sistema de realidad virtual a través de componentes visuales como botones y menús; además de brindar una información visual de lo que ocurre en el ejercicio práctico.

Se pretende que otros simuladores e incluso los juegos puedan tener la opción de brindar este puesto, ya sea para que alguien monitoree el trabajo de otro o para que el propio usuario configure su ejercicio o juego. Por tanto, es conveniente contar con un módulo que automatice la creación de estas interfaces, permitiendo añadir elementos (menús, botones) que serán usados por los clientes para controlar dichos productos.

El proyecto de Herramientas de Desarrollo para Sistemas de Realidad Virtual está realizando una herramienta para la elaboración de SRV. Dicha herramienta no cuenta con un módulo para la creación de interfaces gráficas; por tanto se tiene la necesidad de implementar un Módulo para la creación y control de interfaces gráficas, cuyo resultado, la interfaz generada, pueda ser acoplada a simuladores o juegos de manera fácil para los programadores de los productos finales.

El problema científico a resolver es cómo implementar un Módulo para la creación y control de interfaces visuales para SRV.

2.1.2 Objeto de automatización.

El objeto de automatización del presente proyecto es un Módulo para la creación y control de interfaces visuales para SRV, que no es más que una librería de clases que se encargará de generar interfaces gráficas mediante el uso de llamadas a métodos sencillos declarados e implementados en dichas clases.

2.1.3 Información que se maneja.

La información que se maneja es la referente a los atributos de los componentes panel, menú, botón y etiqueta.

2.1.4 Propuesta de sistema.

El Módulo para la creación y control de interfaces visuales para SRV tendrá una biblioteca de clases diseñada de forma tal que será de fácil acople con la herramienta existente y permitiendo que opere en la base tanto en *OpenGL* como en *DirectX*.

Podrá salvarse y cargarse la interfaz. El usuario podrá crear la interfaz “manualmente” (creando por código los diferentes controles e indicando los parámetros de cada uno) y se permitirá tener una imagen de fondo, la cual se mostrará en un panel. Se programará para esta versión los controles menú, botón, panel y etiqueta al ser estos los más usados en este tipo de interfaces.

Solamente responderán a los eventos del *mouse* los botones y el menú.

Partiendo del principio de que todo lo que se visualizará en la interfaz, que es el resultado esperado por el usuario, son Elementos Visuales, la biblioteca de clases del módulo tendrá una clase controladora de elementos visuales la cual se encargará de definir las características, propiedades y eventos de todos los

componentes visuales en general. De esta clase heredarán las demás clases específicas como son los botones, menús, paneles y etiquetas.

Los controles pueden dar respuestas visuales a las acciones del usuario ante los eventos *OnMouseDown* y *OnMouseMove* (*OnMouseOver*), para lo cual el programador podrá definir las imágenes que mostrarán los controles ante estos eventos.

Se podrá programar los eventos, *OnMouseMove*, *OnMouseDown* de los controles. Para esto se pondrá, para cada componente, estos eventos virtuales, para que el usuario final pueda redefinirlos.

No se dará soporte para esta versión al evento del teclado *OnEnter*.

Se debe permitir la aplicación de transformaciones geométricas como son la rotación y traslación en los distintos ejes de coordenadas, y el escalado de los elementos visuales.

Debe dar la posibilidad al programador de incluir sus propios componentes dando pie de esta manera a actualizaciones futuras.

2.2 Reglas del negocio.

Las dimensiones de las imágenes que se utilizarán para texturizar las mayas, deben ser potencia de dos. El programador debe diseñar estas imágenes de acorde con esta regla. No necesariamente las imágenes deben ser cuadradas.

Las imágenes a cargar deben ser sólo de extensión *.tga* o *.bmp*.

Las imágenes a visualizar con transparencia, deben tener definido su canal alpha.

Los nombres de las imágenes que se cargarán para texturizar las mayas no deben exceder de 16 caracteres, incluida la extensión.

Si se desea visualizar una imagen A encima de otra imagen B, primero se debe visualizar la de fondo (B) y luego la otra (A).

2.3 Modelo de dominio.

A partir del entendimiento del problema se obtiene el siguiente modelo de dominio, el glosario de términos de esta sección se incluye en el glosario de términos general del trabajo. Esto es un pequeño acercamiento a la solución del problema.

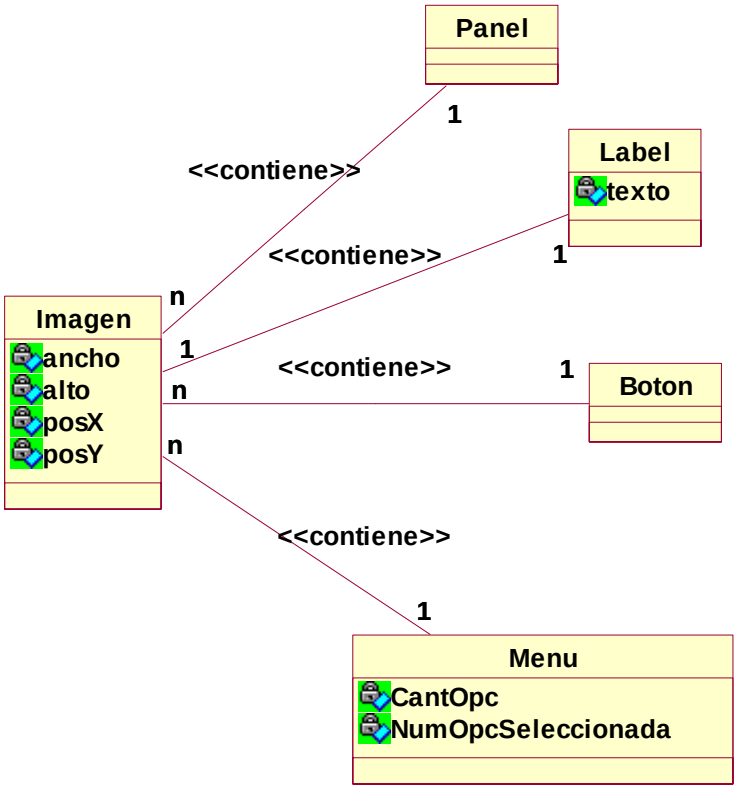

Figura 3 Modelo de dominio.

2.4 Especificación de los requisitos de software.

2.4.1 Dependencias y relaciones con otro software.

Este proyecto esta concebido para que sea un módulo de la herramienta SceneToolKit que se desarrolla en el proyecto Herramientas de Desarrollo de SRV, por lo tanto las clases que formarán parte del Módulo para la creación y control de interfaces gráficas para SRV debe poseer una fuerte interrelación con la biblioteca de clases que conforma dicha herramienta ya que utilizará sus funcionalidades para crear sus propias responsabilidades o métodos.
El módulo de este proyecto respetará la nomenclatura de la herramienta *SceneToolKit* y su estructura de clases.

2.4.2 Requerimientos funcionales.

Teniendo en cuenta las necesidades de los futuros usuarios y la descripción de como debe funcionar el sistema, se pueden inferir los requerimientos funcionales siguientes.

El sistema debe permitir:

1- Salvar la interfaz a un fichero.
2- Cargar la interfaz desde un fichero.
3- Crear botón.
4- Crear menú.
5- Crear panel.
6- Crear etiqueta.
7- Eliminar botón.
8- Eliminar menú.

9- Eliminar panel.
10- Eliminar etiqueta.
11- Modificar botón.
12- Modificar menú.
13- Modificar panel.
14- Modificar etiqueta.
15- Especificar imagen para evento *OnMouseDown* para botón o menú.
16- Especificar imagen para evento *OnMouseOver* para botón o menú.
17- Ante un evento de Mouse (*OnMouseDown*, *Move*, *Up*, *Click*) se debe determinar que componente está siendo afectado.
18- Cambiar la imagen para el evento *OnMouseDown* para botón o menú, si está definida.
19- Cambiar la imagen para el evento *OnMouseOver* para botón o menú, si está definida.

2.4.3 Requerimientos no funcionales.

- Soporte: El sistema deberá correr tanto en *Linux* como en *Windows*.
- Rendimiento: Como aplicación de tiempo real debe tener alto grado de velocidad de procesamiento, tiempo de respuesta y de recuperación.
- Hardware: Compatibilidad con tarjetas gráficas de la familia NVIDIA (*Geforce* 3, *Geforce* 4, FX 5200) al ser estas las especificadas por la mayoría de los usuarios.
- Diseño e Implementación: El módulo debe estar acorde a las características del sistema del cual va a formar parte, por lo que se diseñará siguiendo la filosofía de Programación Orientada a Objetos, programado puramente en C++.
- Uso : Estará dirigida a usuarios con conocimientos de programación general y no necesariamente con conocimientos gráfico; deberá ser usada fácilmente por los mismos por lo que mantendrá los estándares de los entornos de desarrollo de programación visual más conocidos, manteniéndose los eventos estándares.

- Legales: Se regirá por las normas ISO 9000.
- Confiabilidad: Se tendrán en cuenta todas las validaciones para garantizar la seguridad desde el punto de vista de funcionamiento propio del sistema (*crash*) así como ante manipulaciones indebidas de los usuarios.
- Interfaz: El módulo no brindará una interfaz visual pero alejará al usuario de los funcionamientos internos, brindando una interfaz de programación clara con código legible que permita un futuro acceso visual a sus funcionalidades.
- Documentación: Se documentará todo el módulo según los estándares de documentación establecidos en el proyecto.

2.5 Definición de los casos de uso del sistema.

A continuación se reconocen los actores del sistema a desarrollar y se conciben, a partir de los requisitos funcionales anteriormente hallados, los casos de uso del sistema.

Además, se seleccionan los casos de uso correspondientes al primer ciclo de desarrollo para hacerles sus especificaciones textuales en formato expandido.

2.5.1 Actor del sistema

Actores	**Justificación**
Programador	Las clases de este módulo estarán insertadas junto con las clases de la herramienta *SceneToolKit* y cuya finalidad será el uso que pueda hacer de ellas los programadores, por lo tanto son los que se beneficiarán con las funcionalidades que brinda el módulo.

Tabla 5 Justificación de actores.

2.5.2 Casos de uso de sistema

CU-1	Salvar y cargar interfaz.
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador manda a salvar o a cargar la interfaz y finaliza cuando esta se encuentra salvada o cargada.
Referencia	R1 y R2

Tabla 6 CU Cargar interfaz desde un fichero

CU-2	Crear control
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador indica la creación de un control y termina cuando esta acción ha sido llevada a cabo.
Referencia	R3, R4, R5 y R6

Tabla 7 CU Crear control

CU-3	Modificar control
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador indica la modificación de un control y termina cuando esta acción ha sido llevada a cabo.
Referencia	R11, R12, R13 y R14

Tabla 8 CU Modificar control

CU-4	Eliminar control
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador indica la eliminación de un control y termina cuando esta acción ha sido llevada a cabo.
Referencia	R7, R8, R9 y R10

Tabla 9 CU Eliminar control

CU-5	Responder a eventos del *mouse.*
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador indica dar respuesta a un evento del *mouse* y termina cuando esta acción ha sido llevada a cabo.
Referencia	R15, R16, R17, R18 y R19

Tabla 10 CU Responder a eventos del *mouse*.

CU-6	Determinar componente afectado.
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador manda a determinar cual componente que ha sido afectado por algún evento del *mouse* y termina cuando se conoce este componente.
Referencia	R17

Tabla 11 CU Determinar componente afectado

CU-7	Actualizar imagen del componente.
Actor	Programador
Descripción	El caso de uso se inicia cuando el programador indica la actualización de la imagen del componente y termina cuando esta acción ha sido llevada a cabo.
Referencia	R15, R16, R18 y R19

Tabla 12 CU Actualizar imagen del componente

2.5.3 Diagrama de actores y casos de uso del sistema.

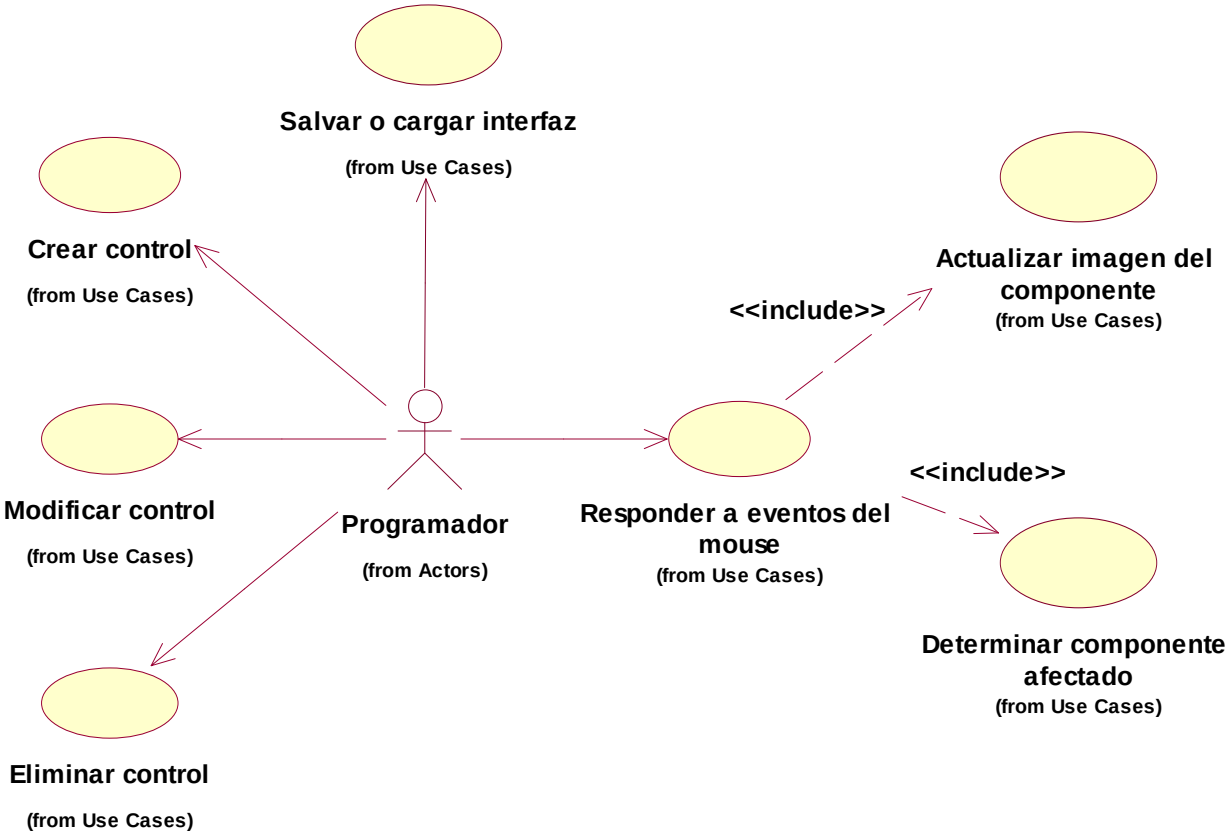

Figura 4 Diagrama de casos de uso del sistema.

2.5.4 Expansión de los casos de uso del sistema.

2.5.4.1 CU Salvar o cargar interfaz.

Caso de uso		
CU-1	Salvar o cargar interfaz.	
Propósito	Permitir al usuario final que desde un fichero pueda cargar la interfaz ya definida por el programador.	
Actores: Programador (inicia)		
Resumen: El caso de uso se inicia cuando el programador manda a salvar la interfaz en un fichero o cuando manda a cargar la interfaz desde un fichero y finaliza cuando esta se encuentra salvada o cargada respectivamente.		
Referencias	R1, R2	
Acción del actor		**Respuesta del sistema**
1. El programador manda a salvar o a cargar la interfaz.		1.1 El sistema salva o carga la interfaz.
Precondiciones	-	
Poscondiciones	Se efectuó la carga de la interfaz grafica con todos los componentes definidos por el programador desde un fichero o se efectúa la salva de la interfaz con los valores de todos sus componentes en un fichero.	

Tabla 13 Expansión del CU Salvar o cargar interfaz.

2.5.4.2 CU Crear control.

<table>
<tr><th colspan="3">Caso de uso</th></tr>
<tr><td>CU-2</td><td colspan="2">Crear control</td></tr>
<tr><td>Propósito</td><td colspan="2">Crear un control</td></tr>
<tr><td colspan="3">Actores: Programador (inicia).</td></tr>
<tr><td colspan="3">Resumen: El caso de uso se inicia cuando el programador indica la creación de un control y termina cuando el control fue creado.</td></tr>
<tr><td>Referencias</td><td colspan="2">R3, R4, R5 y R6.</td></tr>
<tr><td colspan="3">Flujo Normal de los Eventos</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2">1. El programador manda a crear un control ya sea un menú, panel, botón o etiqueta.</td><td>1.1 En el caso de que el control que se desee crear sea:

a) Botón, ir a la sección “Crear Control Imagen”.
b) Panel, ir a la sección “Crear Control Imagen”.
c) Menú, ir a la sección “Crear Menú”.
d) Etiqueta, ir a la sección “Crear Etiqueta”.</td></tr>
<tr><td colspan="3">Sección “Crear Control Imagen”</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2">2. El programador manda a cargar las imágenes que se le asociaran al control.</td><td>2.1 El sistema carga las imágenes si el camino dado por el programador es el correcto.</td></tr>
<tr><td colspan="2">3. El programador crea un objeto Control Imagen.</td><td>3.1 El sistema crea el correspondiente nodo de geometría asignándoles las imágenes creadas como texturas y adicionándolo como hijo al grupo de nodos de la escena.</td></tr>
</table>

Sección "Crear Menú"	
Acción del actor	**Respuesta del sistema**
2. El programador manda a cargar las imágenes que se le asociaran al control.	2.1 El sistema carga las imágenes si el camino dado por el programador es el correcto.
3. El programador crea un objeto Menú.	3.1 El sistema crea el correspondiente grupo de nodos del menú adicionándolo como hijo al grupo de nodos del la escena.
4. El programador manda a crear una opción del menú.	4.1 El sistema crea el correspondiente nodo de geometría asignándoles las imágenes creadas como texturas y adicionándolo como hijo al grupo de nodos del menú.
Sección "Crear Etiqueta"	
Acción del actor	**Respuesta del sistema**
2. El programador introduce el texto, el color, la posición y la cámara donde desea que la etiqueta se visualice.	2.1 El sistema crea la etiqueta de acuerdo a los parámetros entrados por el programador.
Precondiciones	-
Poscondiciones	Se efectúan las correspondientes acciones sobre un control.

Tabla 14 Expansión del CU Crear control.

2.5.4.3 CU Modificar control.

Caso de uso	
CU-3	Modificar control
Propósito	Modificar un control.
Actores: Programador (inicia).	
Resumen: El caso de uso se inicia cuando el programador indica la modificación de un control y termina cuando esta acción ha sido llevada a cabo.	
Referencias	R11, R12, R13 y R14.
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1. El programador manda a modificar un parámetro de un control, ya sea este último un menú, panel, botón o etiqueta.	1.1 El sistema modifica el parámetro de acuerdo al valor proporcionado por el programador.
Precondiciones	-
Poscondiciones	Se efectúan las correspondientes acciones sobre un control.

Tabla 15 Expansión del CU Modificar control.

2.5.4.4 CU Eliminar control.

Caso de uso	
CU-4	Eliminar control
Propósito	Eliminar un control
Actores: Programador (inicia).	
Resumen: El caso de uso se inicia cuando el programador indica la eliminación de un control y termina el control ha sido eliminado.	
Referencias	R7, R8, R9 y R10.
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1. El programador manda a eliminar un control.	1.1 El sistema elimina el control quitándolo de su correspondiente grupo de nodos.
Precondiciones	-
Poscondiciones	Se efectúan las correspondientes acciones sobre un control.

Tabla 16 Expansión del CU Eliminar control.

2.5.4.5 CU Responder a eventos del *mouse*.

Caso de uso	
CU-5	Responder a eventos del *mouse*.
Propósito	Dar una respuesta visual a los eventos *OnMouseMove* y *OnMouseDown* por parte de los componentes.
Actores: Programador (inicia).	
Resumen: El caso de uso se inicia cuando el programador indica dar respuesta a los eventos *OnMouseMove* y *OnMouseDown*, y termina cuando esta acción ha sido llevada a cabo.	
Referencias	R15, R16, R17, R18 y R19
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1. El programador indica la captura del clic del mouse.	1.1 El sistema llama al caso de uso incluido “Determinar componente afectado”. 1.2 El sistema llama al caso de uso incluido “Actualizar imagen del componente”.
Precondiciones	-
Poscondiciones	La respuesta visual por parte de los componentes a sido llevada a cabo.

Tabla 17 Expansión del CU Responder a eventos del *mouse*

2.5.4.6 CU Determinar componente afectado.

Caso de uso	
CU-6	Determinar componente afectado.
Propósito	Saber que componente ha sido afectado por algún evento del *mouse*.
Actores: Programador (inicia).	
Resumen: El caso de uso se inicia cuando el programador manda a determinar cual componente que ha sido afectado por algún evento del *mouse* y termina cuando se conoce este componente.	
Referencias	R17
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1. Se manda a determinar si un componente ha sido afectado por el *mouse* (si el mouse se encuentra sobre el componente o si le ha dado un clic).	1.1 El sistema devuelve verdadero si el *mouse* esta sobre el componente y falso en caso contrario.
Precondiciones	-
Poscondiciones	Se conoce que componente ha sido afectado por el *mouse*.

Tabla 18 Expansión del CU Determinar componente afectado

2.5.4.7 CU Actualizar imagen del componente.

Caso de uso	
CU-7	Actualizar imagen del componente.
Propósito	Al darse algún evento del *mouse* sobre algún componente botón o menú, actualizar la imagen del componente.
Actores: Programador (inicia).	
Resumen: El caso de uso se inicia cuando el programador indica la actualización de la imagen del componente, y termina cuando esta acción ha sido llevada a cabo.	
Referencias	R15, R16, R18 y R19
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1. Se indica la actualización de la imagen del componente.	1.1 El sistema actualiza la imagen del componente.
Precondiciones	-
Poscondiciones	La imagen del componente ha sido actualizada.

Tabla 19 Expansión del CU Actualizar imagen del componente.

Conclusiones

En este capítulo se recopilaron los requisitos funcionales y no funcionales y se definieron y describieron los casos de uso de acuerdo con lo que espera el cliente que realice el sistema. Ya se puede proceder con el diseño del sistema.

Capítulo 3 Diseño del sistema.

Introducción

En este capitulo se describen todos los aspectos del sistema a construir mediante el diagrama de clases y los diagramas de secuencia de la realización de los casos de uso, que intervendrán en el primer ciclo de desarrollo del proyecto. Este paso constituye el refinamiento de las etapas anteriores.

3.1 Diagramas de secuencia.

A continuación se presentan los diagramas de secuencia agrupados por paquetes de casos de uso.

3.1.1 Realización del caso de uso “Responder a eventos del mouse”

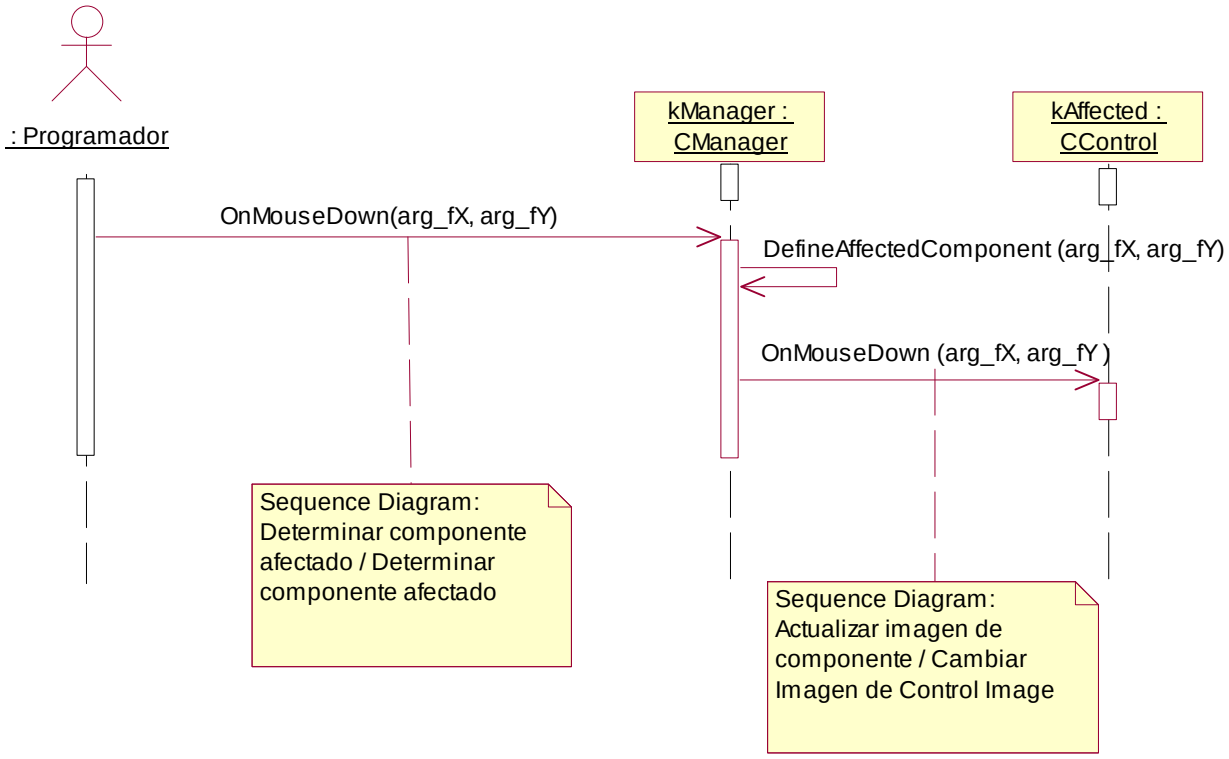

Figura 5 Diagrama de secuencia “Responder a eventos del *mouse*”.

3.1.2 Realización del caso de uso “Determinar componente afectado”.

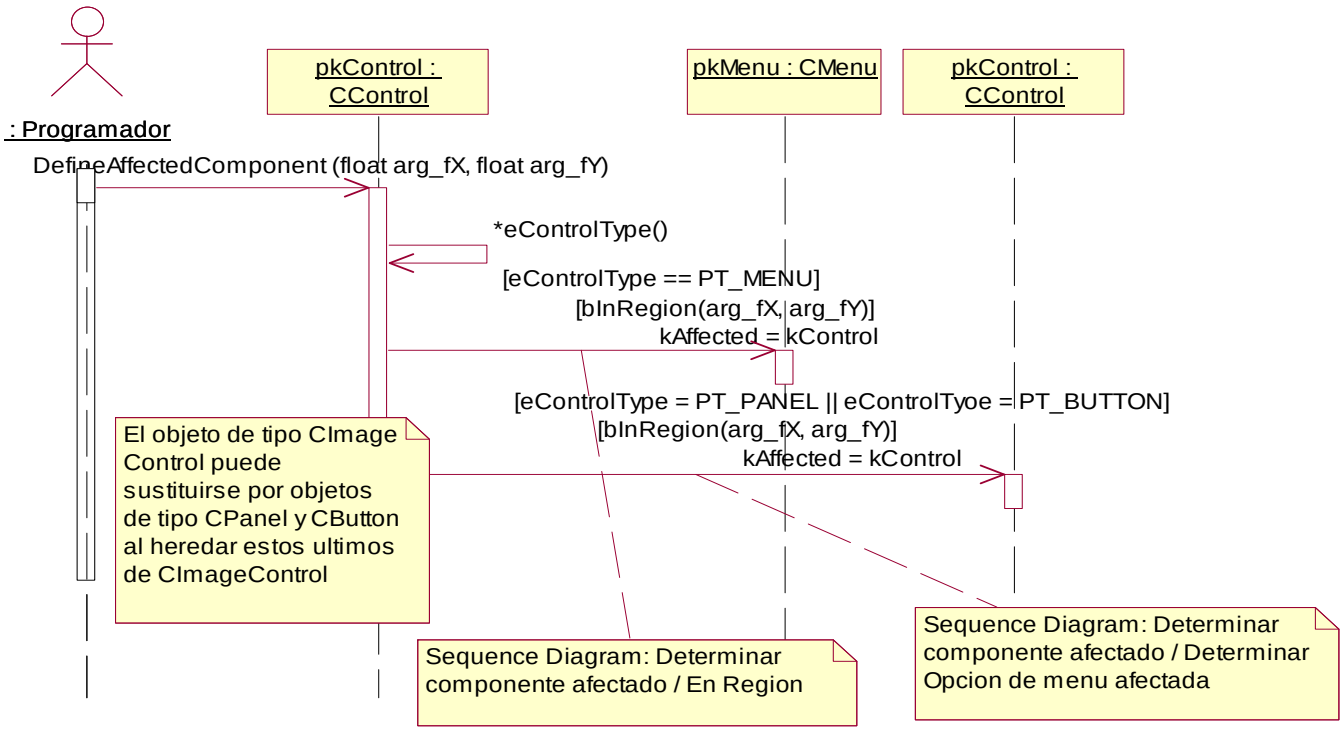

Figura 6 Diagrama de secuencia “Determinar componente afectado”.

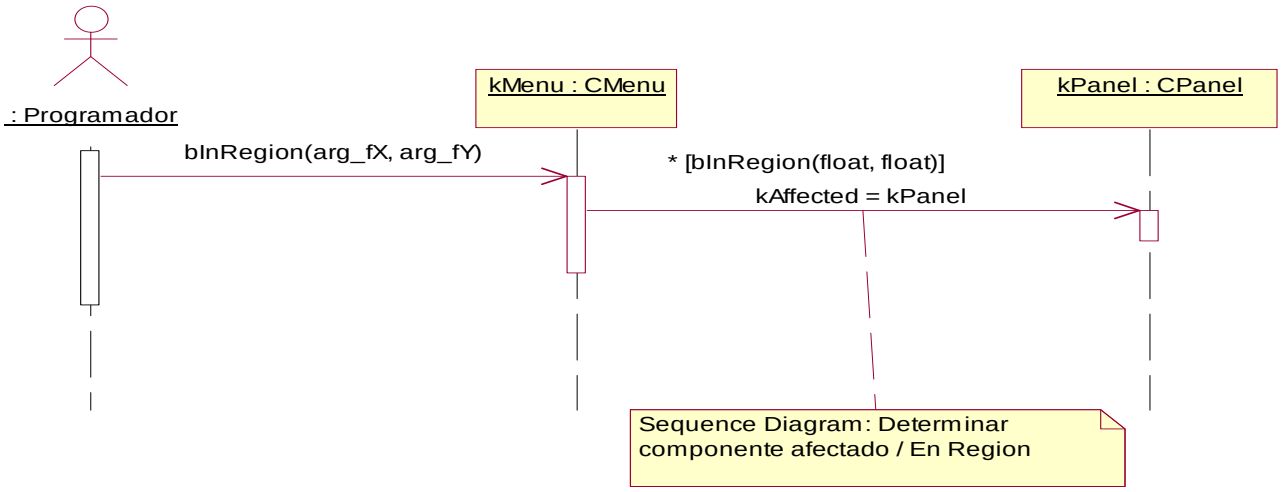

Figura 7 Diagrama de secuencia “Determinar opción de menú afectada”

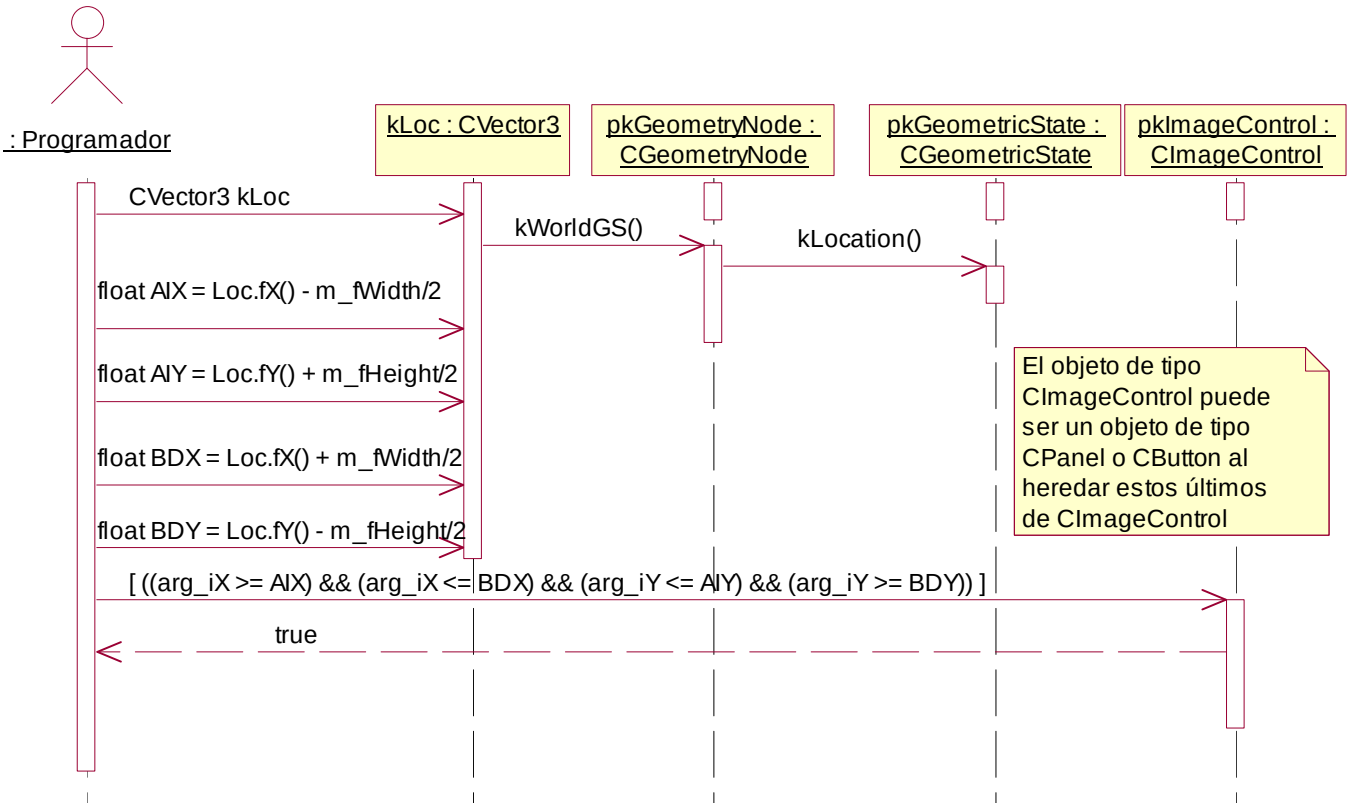

Figura 8 Diagrama de secuencia "¿En Región?"

3.1.3 Realización del caso de uso “Actualizar imagen de componente”.

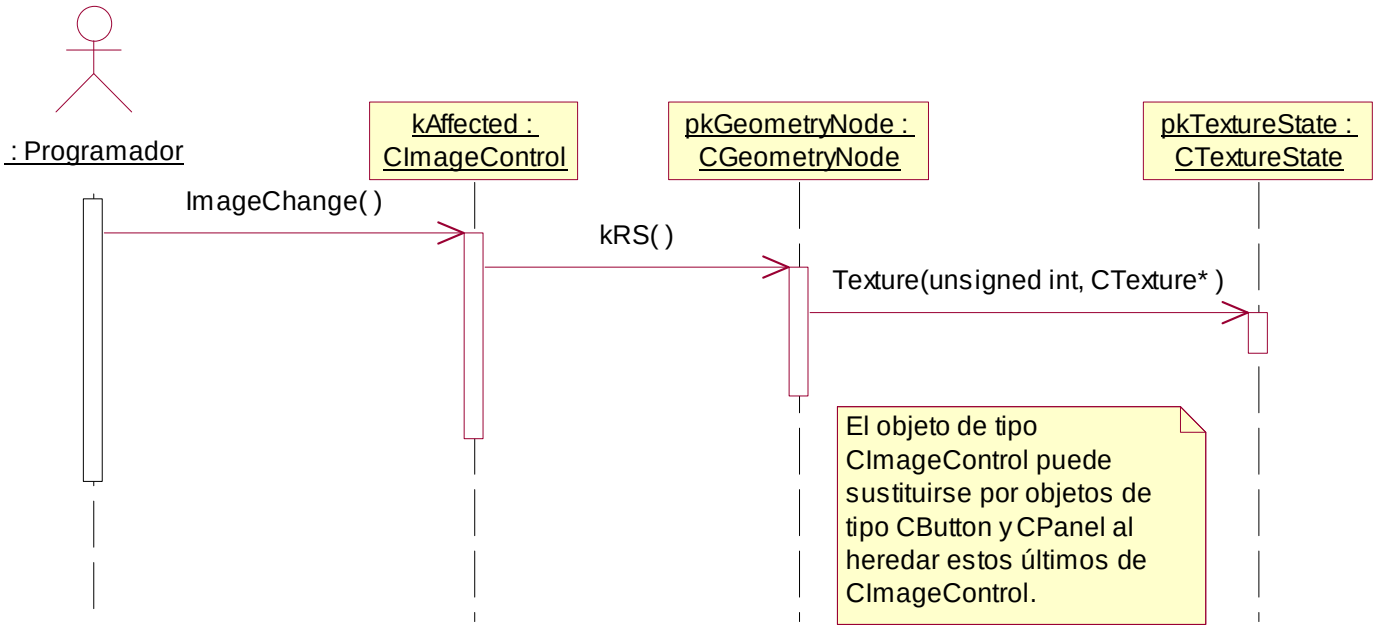

Figura 9 Diagrama de secuencia “Actualizar imagen de componente”.

3.1.4 Realización del caso de uso “Crear control”

En este paquete, el objeto de tipo *CImageControl* puede sustituirse por objetos de tipo *CButton* y *CPanel* al heredar estos últimos de *CImageControl*.

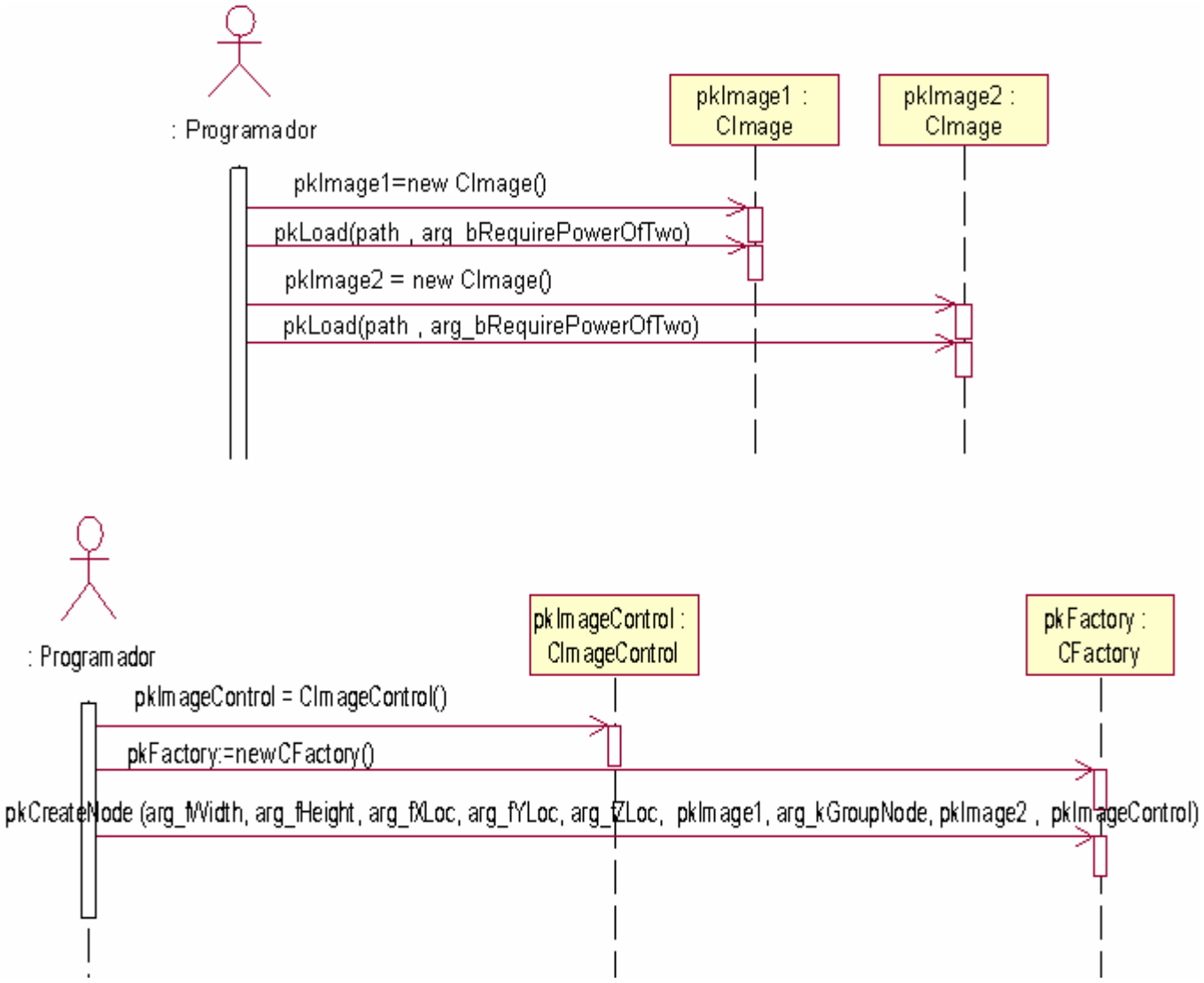

Figura 10 Diagrama de secuencia “Crear control imagen”

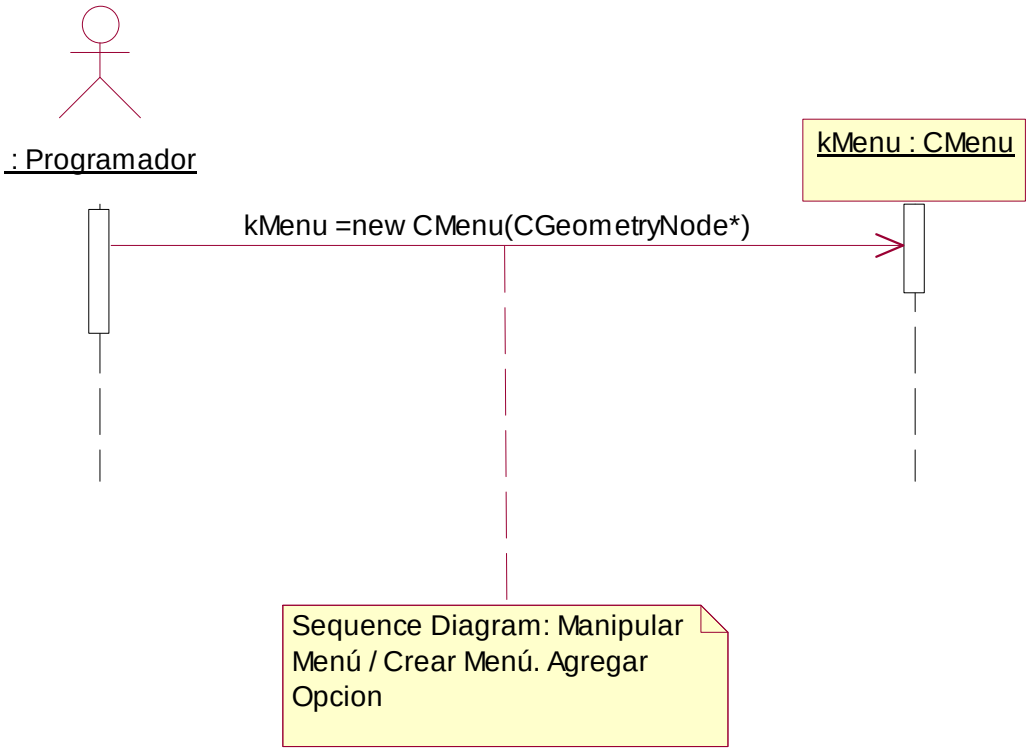

Figura 11 Diagrama de secuencia "Crear menú".

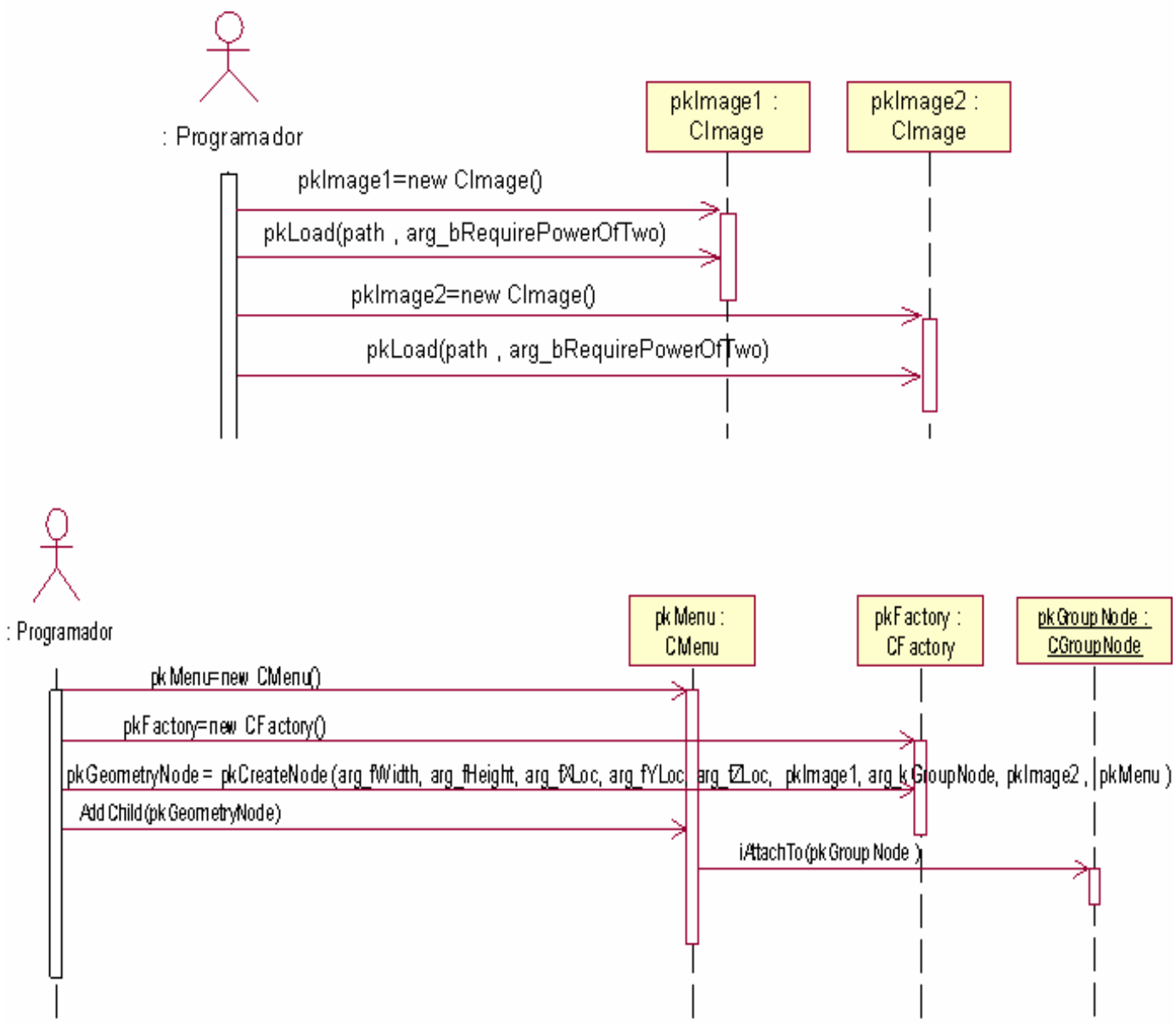

Figura 12 Diagrama de secuencia "Crear menú. Agregar opción".

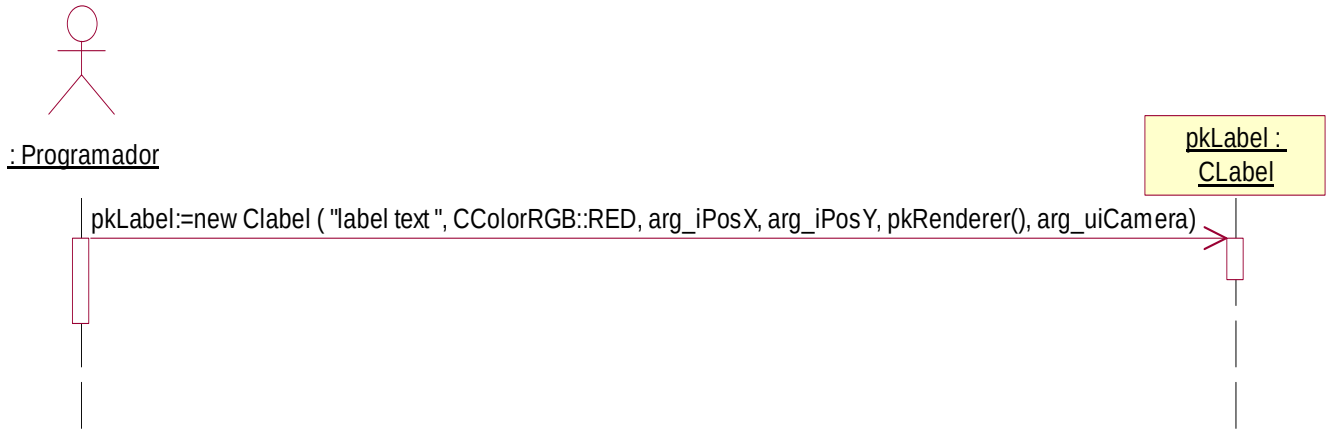

Figura 13 Diagrama de secuencia "Crear etiqueta".

3.1.5 Realización del caso de uso "Modificar control"

En este paquete, el objeto de tipo *CImageControl* puede sustituirse por objetos de tipo *CButton* y *CPanel* al heredar estos últimos de *CImageControl*.

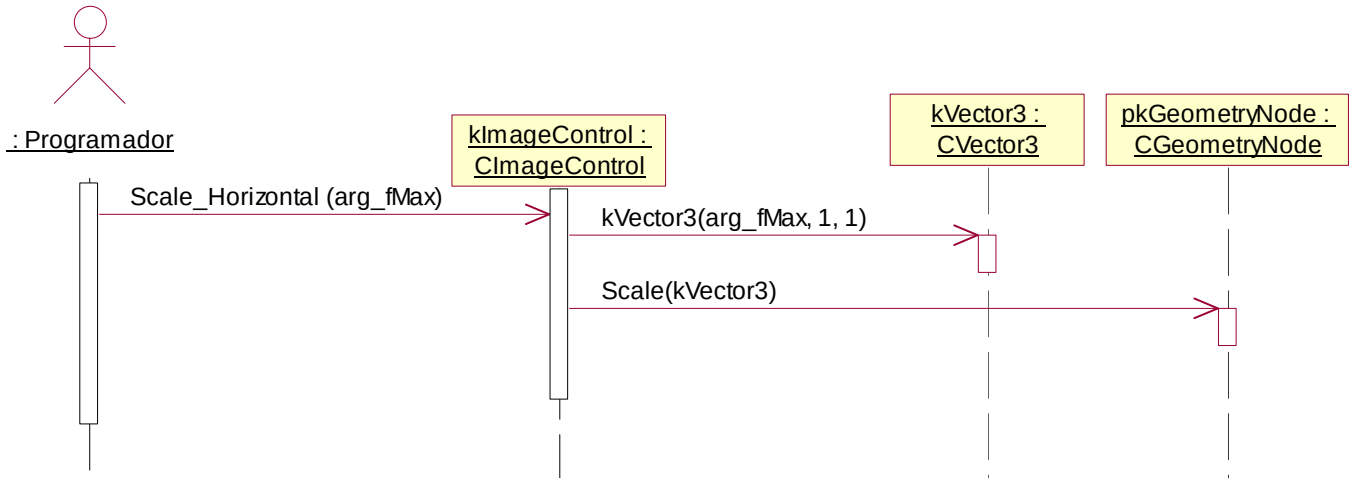

Figura 14 Diagrama de secuencia "Escalar horizontal un control imagen".

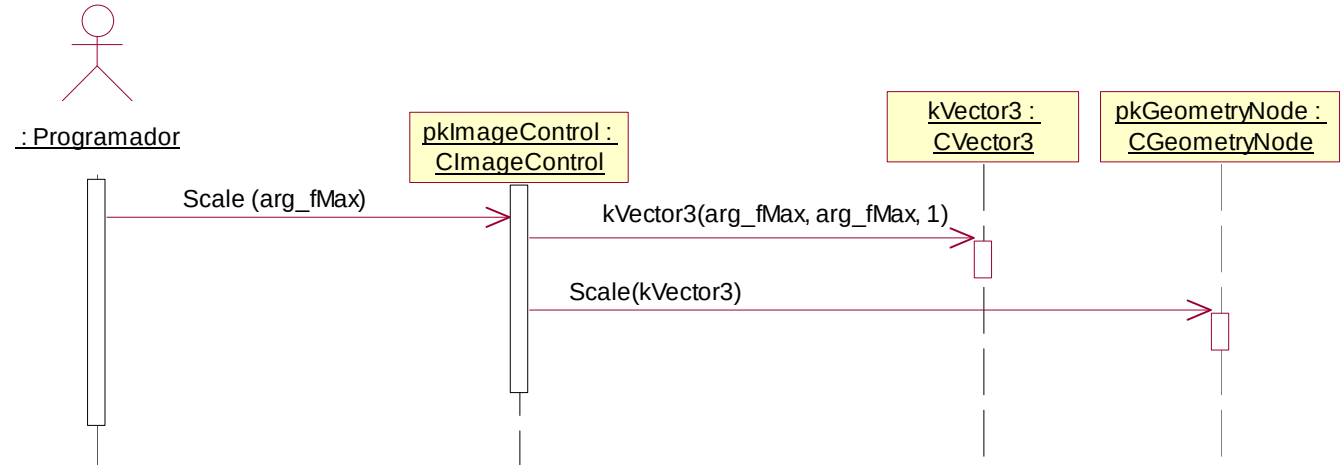

Figura 15 Diagrama de secuencia "Escalar un control imagen en ambos ejes".

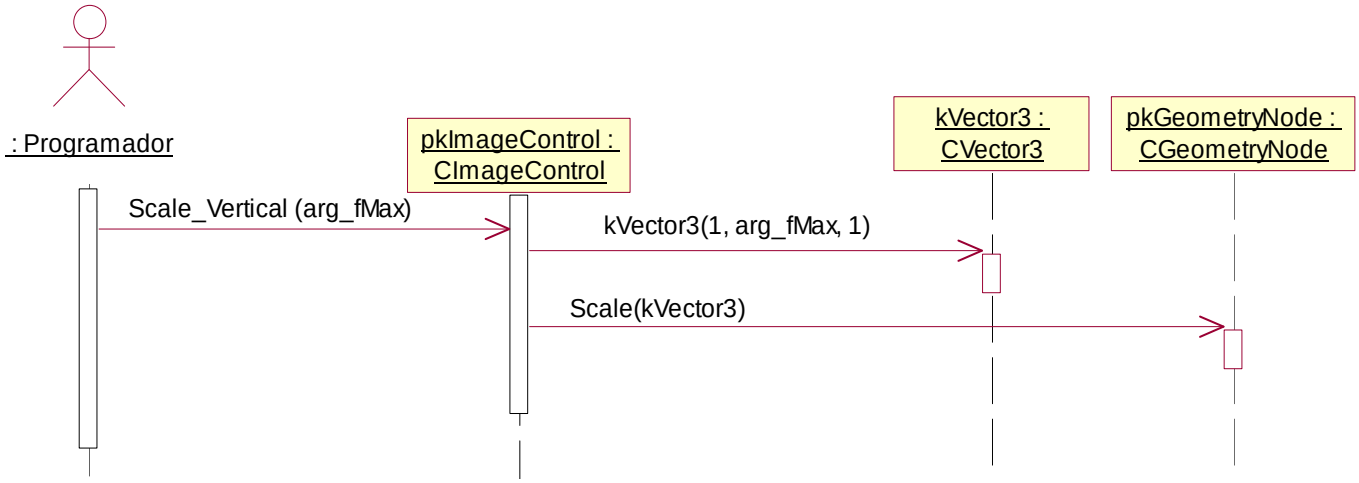

Figura 16 Diagrama de secuencia "Escalar vertical un control imagen".

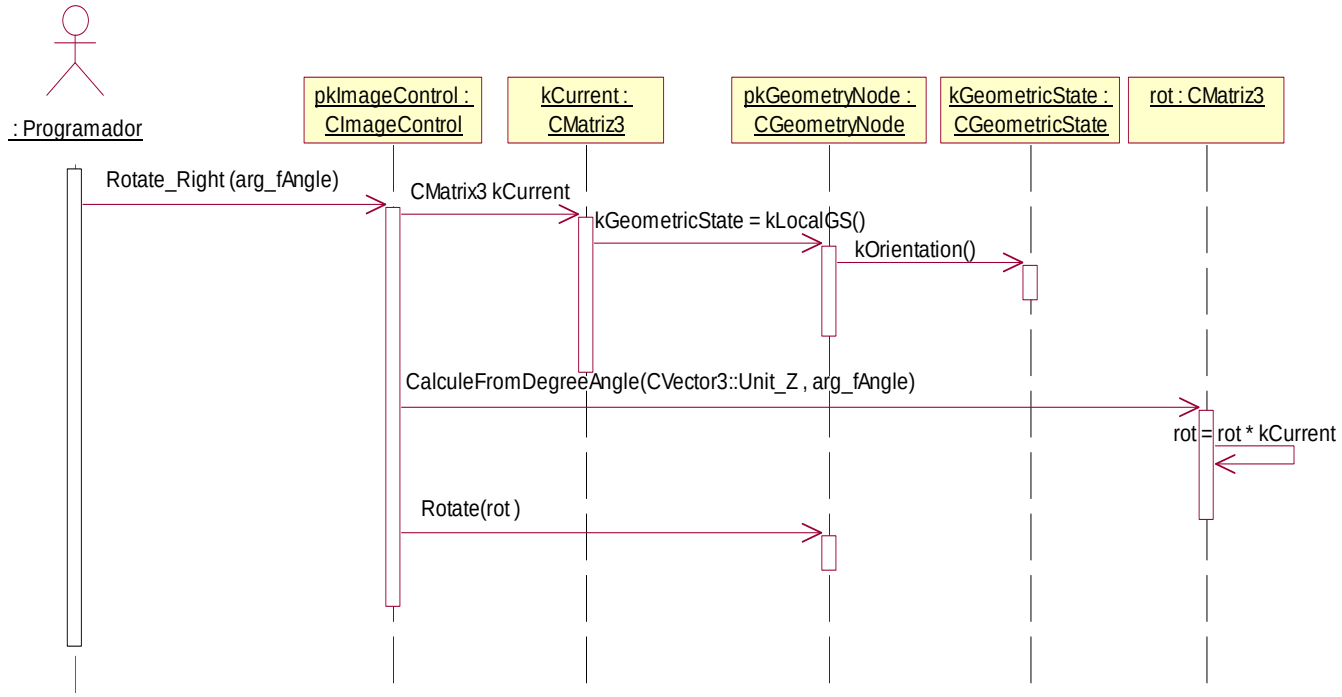

Figura 17 Diagrama de secuencia "Rotar un control imagen a la derecha".

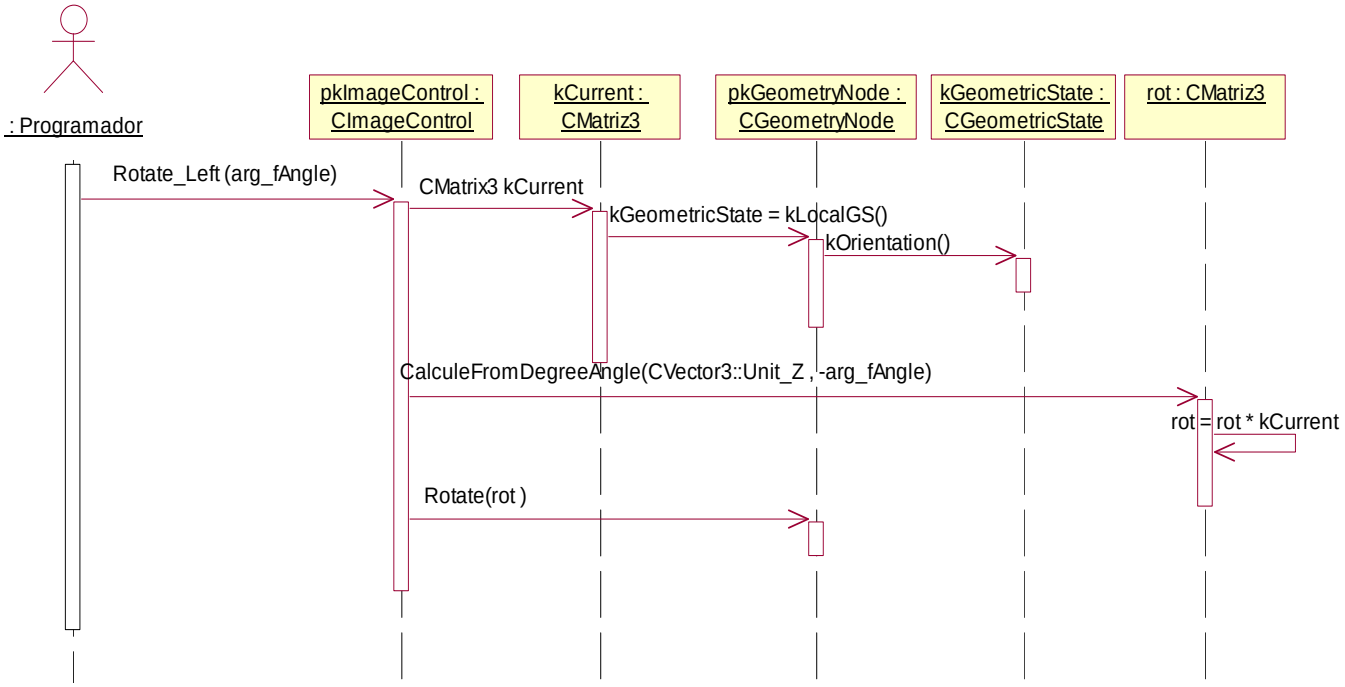

Figura 18 Diagrama de secuencia "Rotar un control imagen a la izquierda".

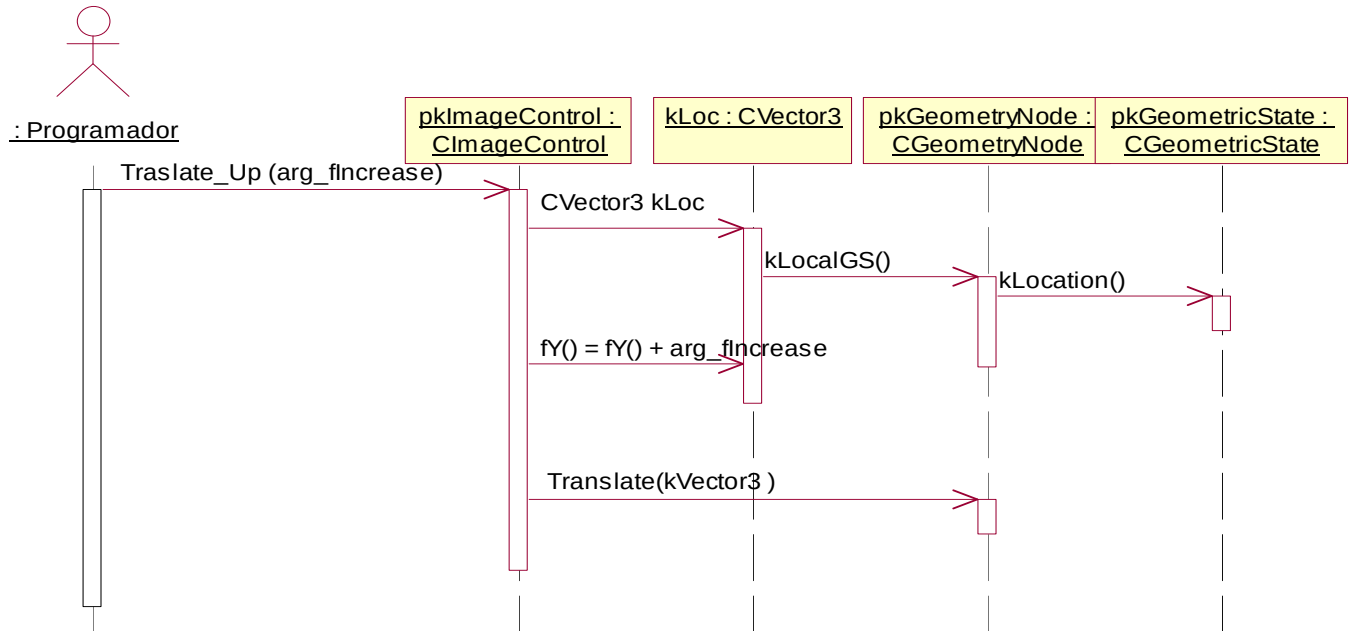

Figura 19 Diagrama de secuencia. "Trasladar un control imagen hacia arriba".

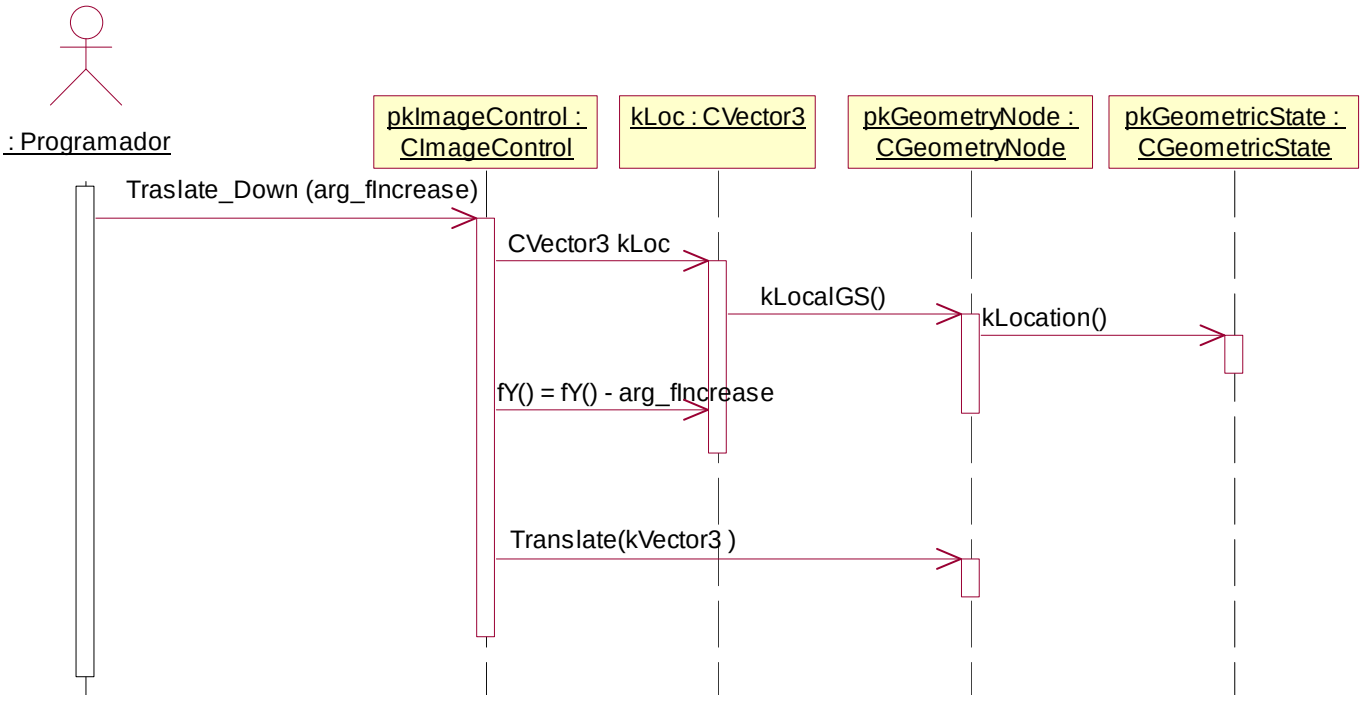

Figura 20 Diagrama de secuencia. "Trasladar control imagen hacia abajo".

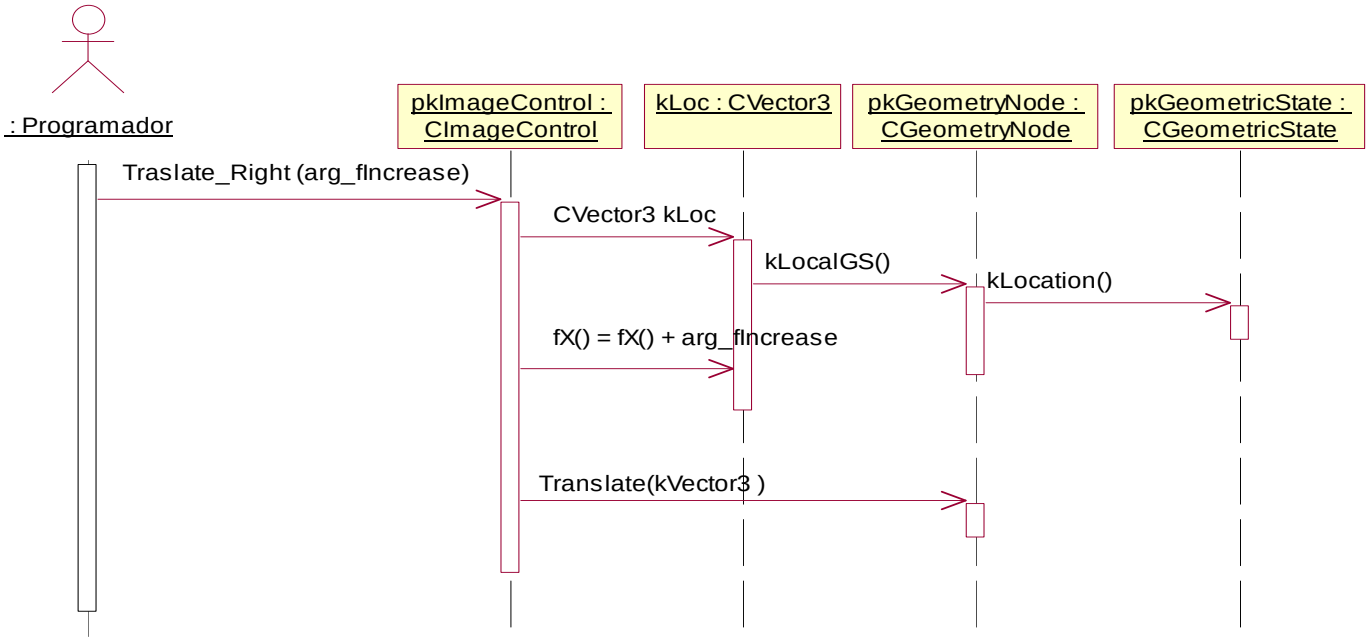

Figura 21 Diagrama de secuencia. "Trasladar un control imagen hacia la derecha".

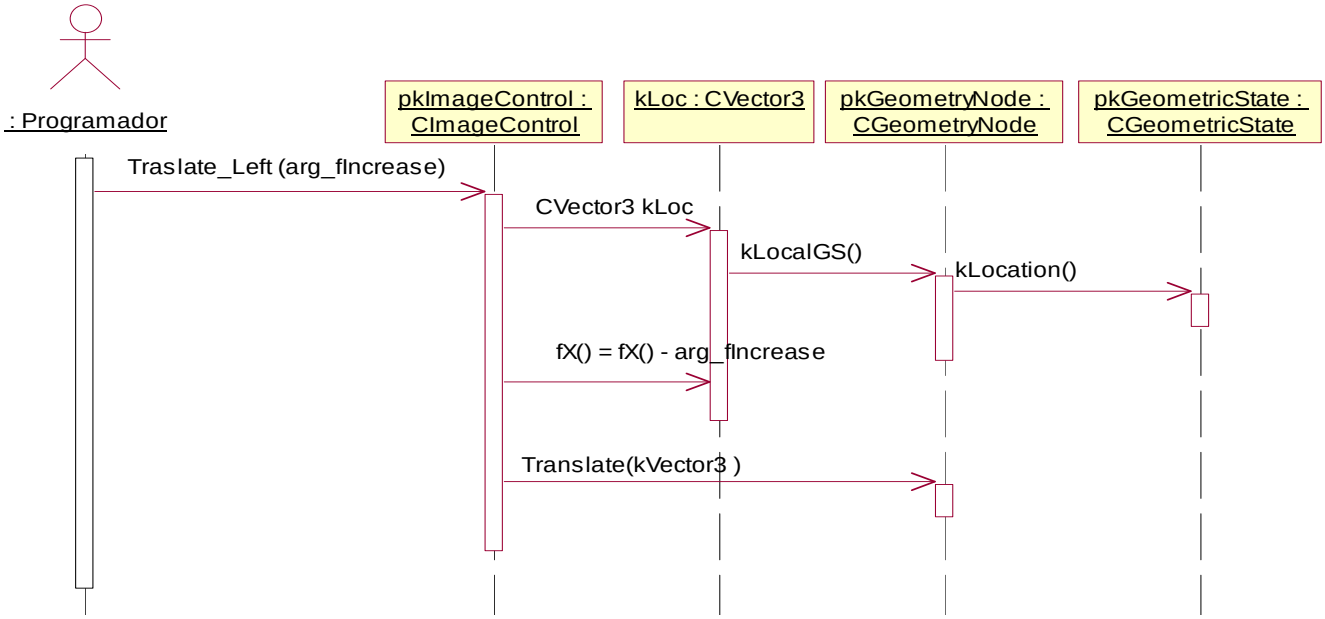

Figura 22 Diagrama de secuencia. "Trasladar un control imagen hacia la izquierda".

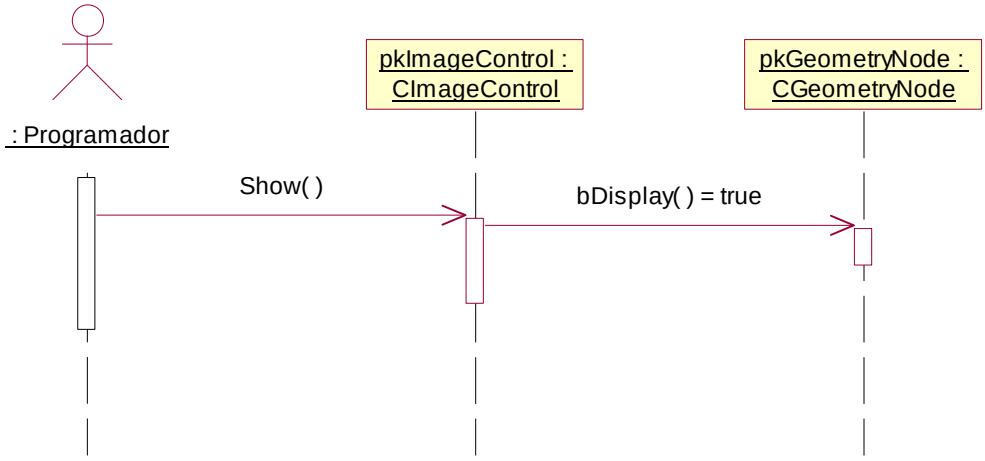

Figura 23 Diagrama de secuencia. "Mostrar control imagen".

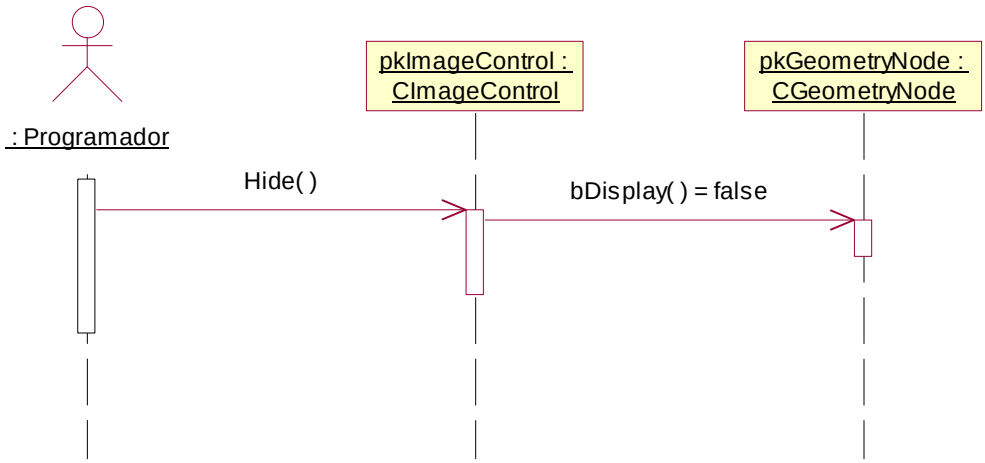

Figura 24 Diagrama de secuencia. "Ocultar control imagen"

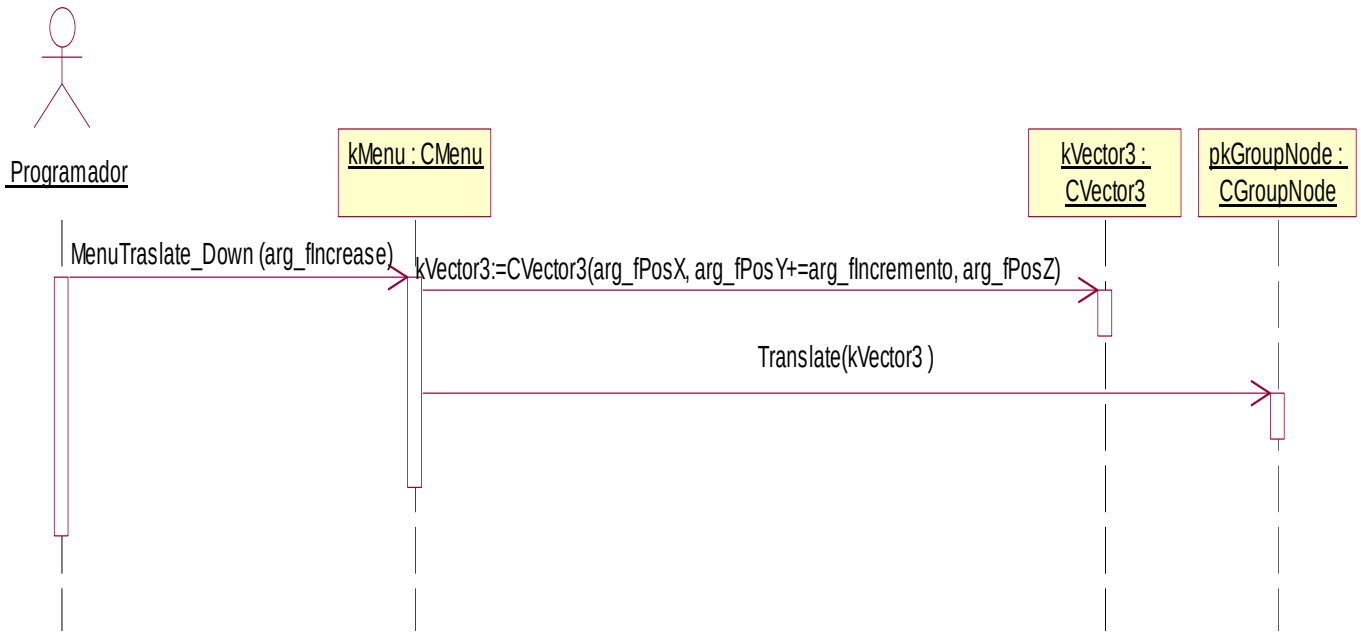

Figura 25 Diagrama de secuencia "Trasladar menú hacia abajo".

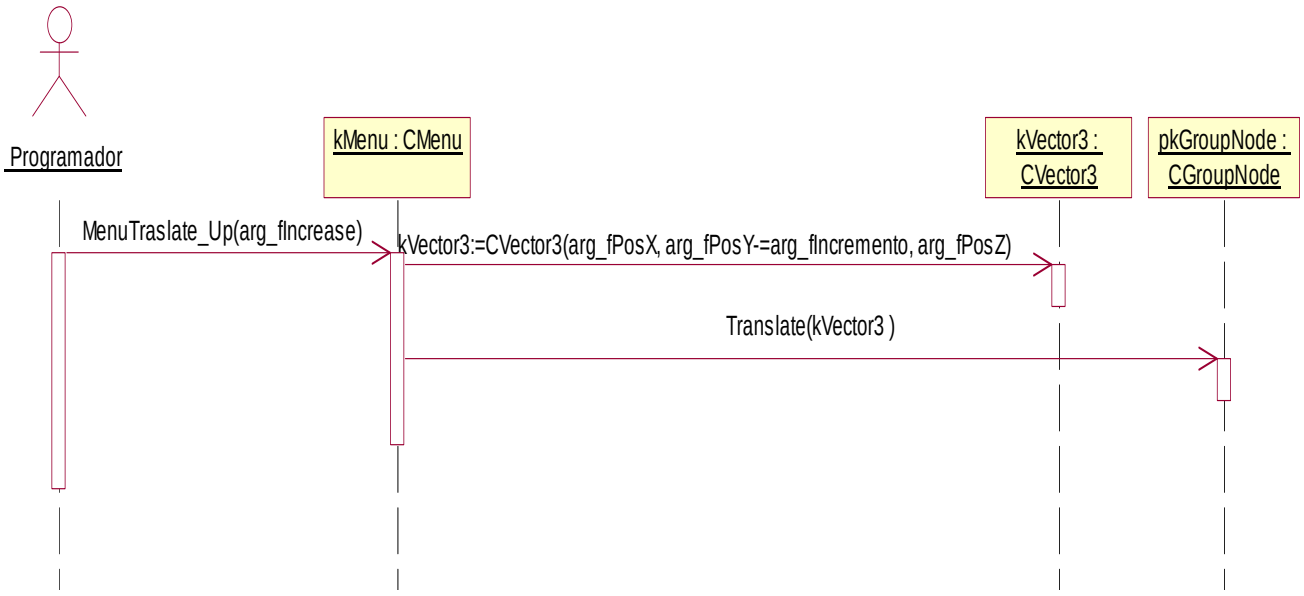

Figura 26 Diagrama de secuencia "Trasladar menú hacia arriba".

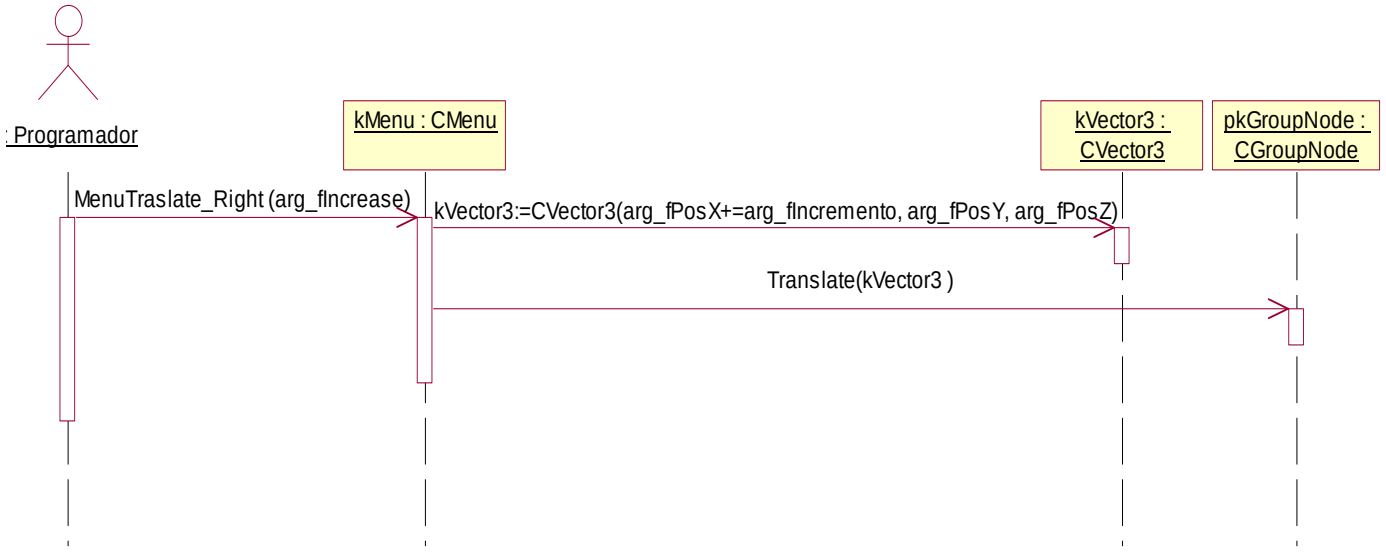

Figura 27 Diagrama de secuencia "Trasladar menú hacia la derecha".

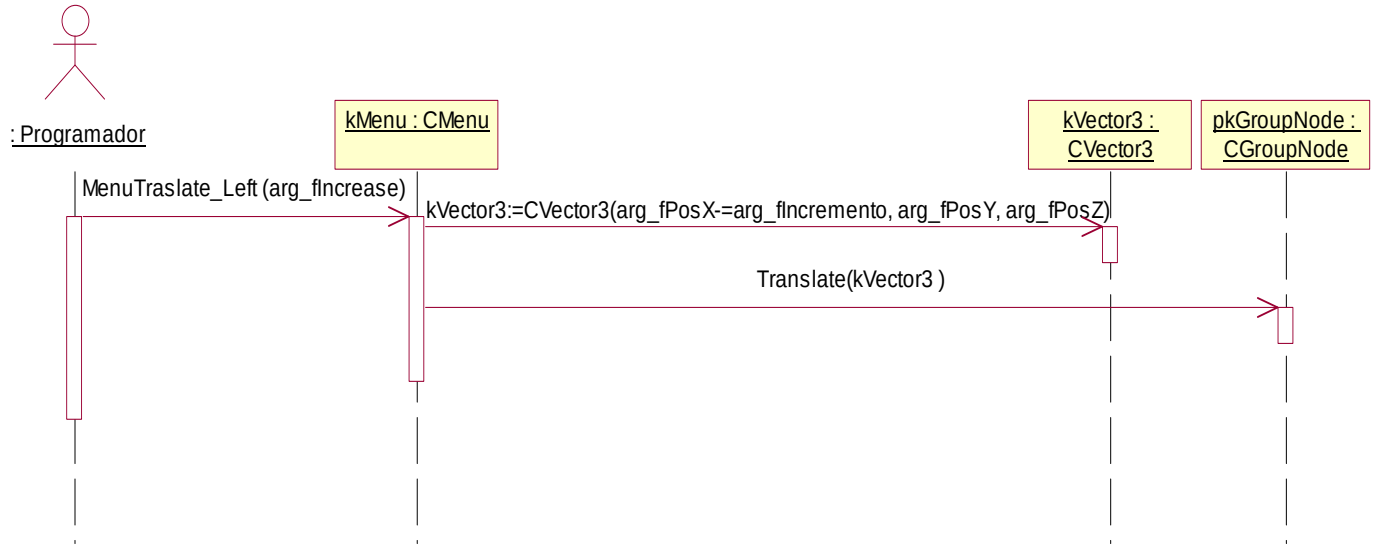

Figura 28 Diagrama de secuencia "Trasladar menú hacia la izquierda".

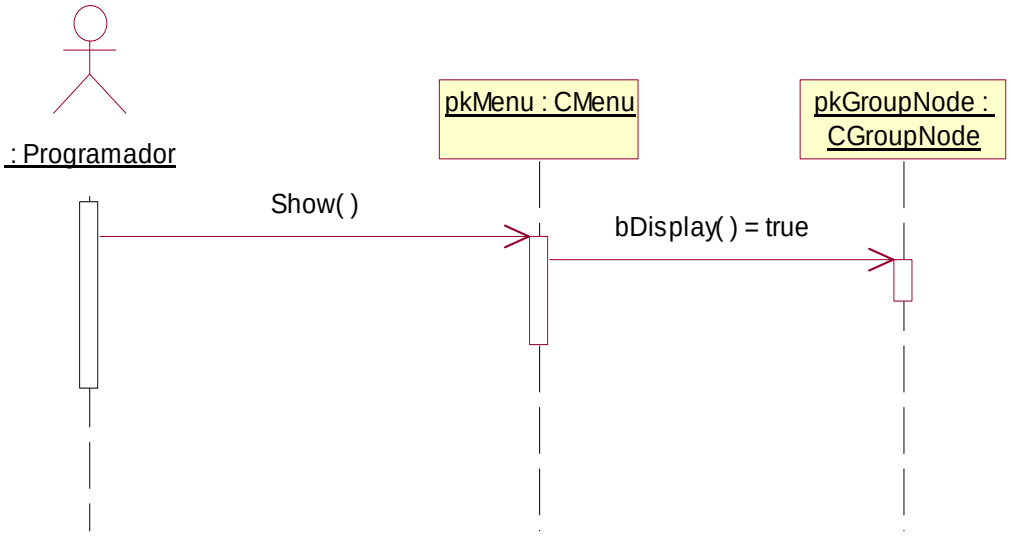

Figura 29 Diagrama de secuencia "Mostrar menú".

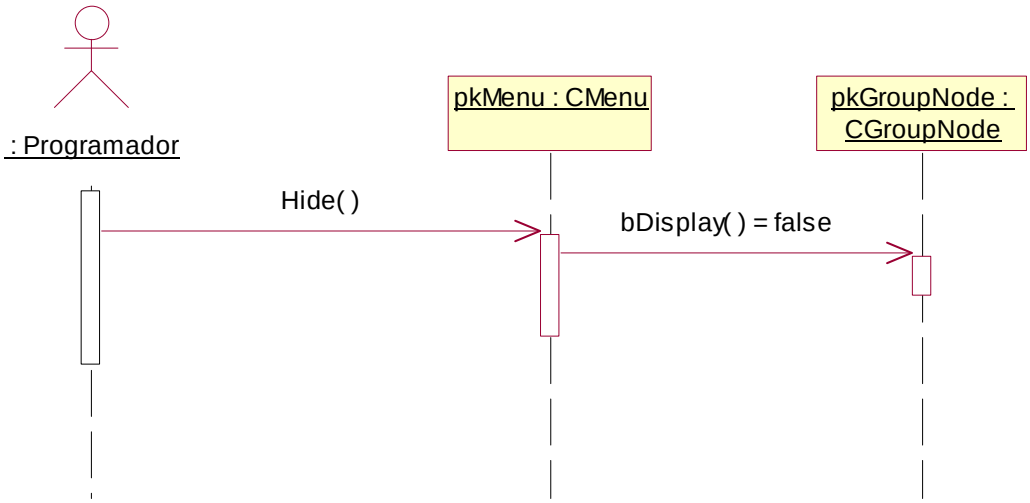

Figura 30 Diagrama de secuencia "Ocultar menú".

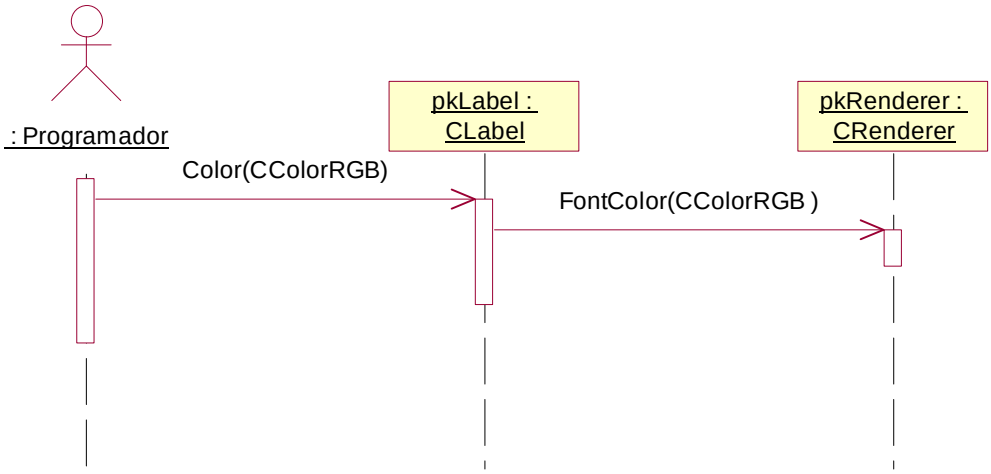

Figura 31 Diagrama de secuencia "Modificar etiqueta. Color".

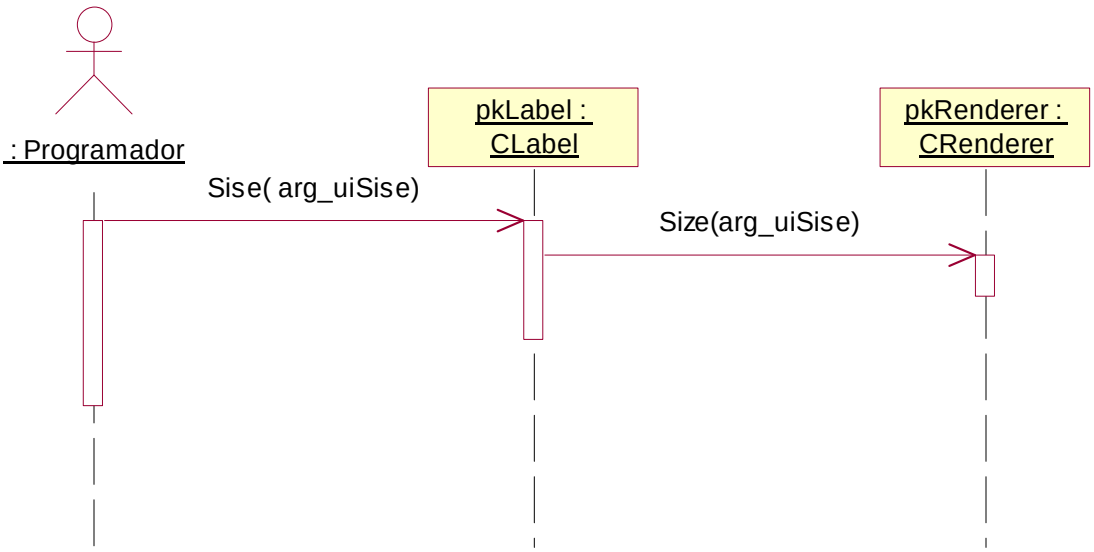

Figura 32 Diagrama de secuencia "Modificar etiqueta. Tamaño".

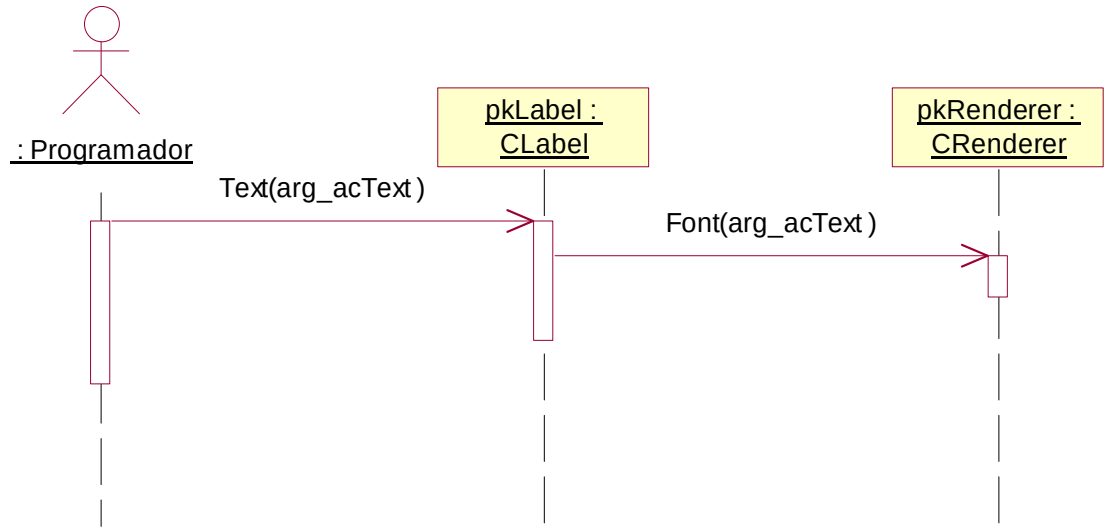

Figura 33 Diagrama de secuencia "Modificar etiqueta. Texto".

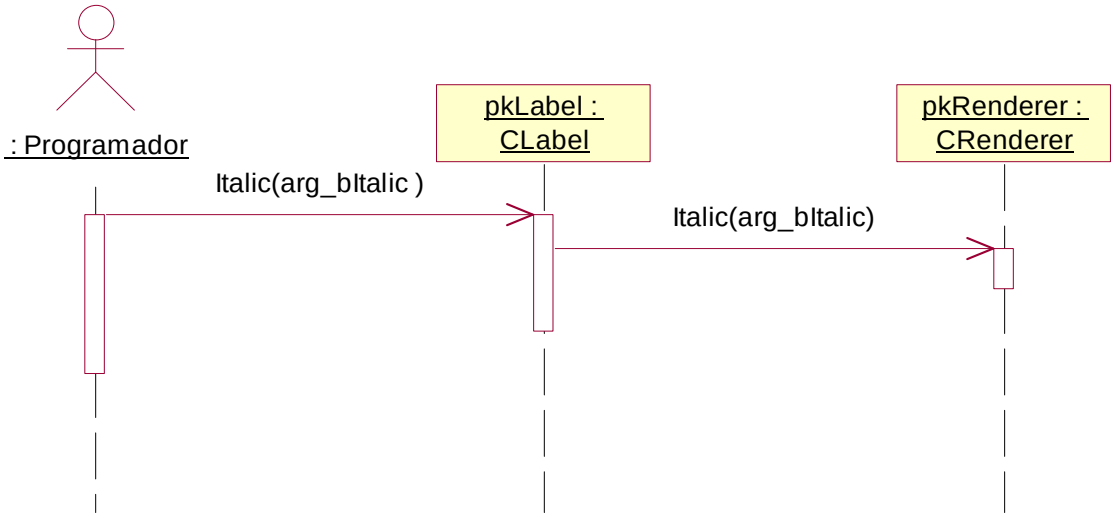

Figura 34 Diagrama de secuencia "Modificar etiqueta. *Italic*".

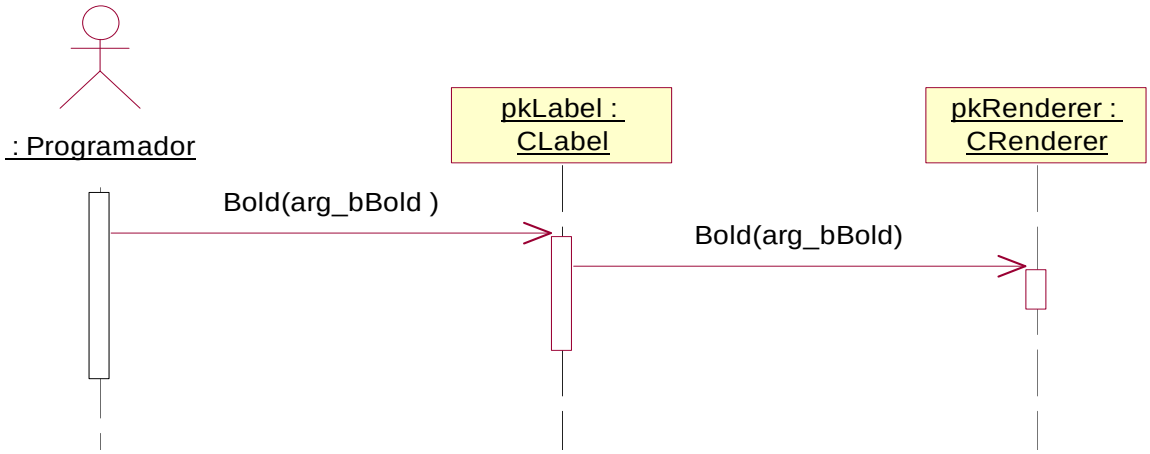

Figura 35 Diagrama de secuencia "Modificar etiqueta. *Bold*".

3.1.6 Realización del caso de uso "Eliminar control"

En este paquete, el objeto de tipo *CImageControl* puede sustituirse por objetos de tipo *CButton* y *CPanel* al heredar estos últimos de *CImageControl*.

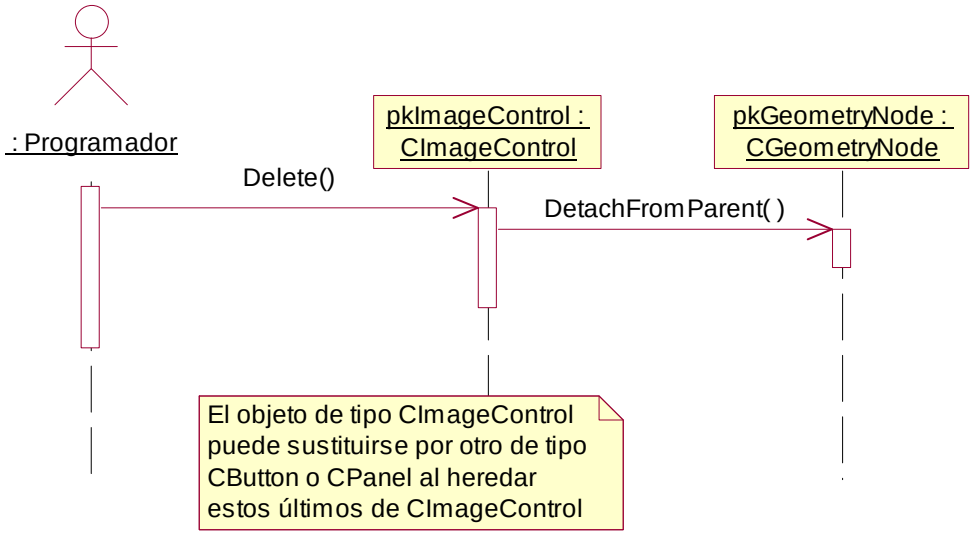

Figura 36 Diagrama de secuencia. "Eliminar control imagen."

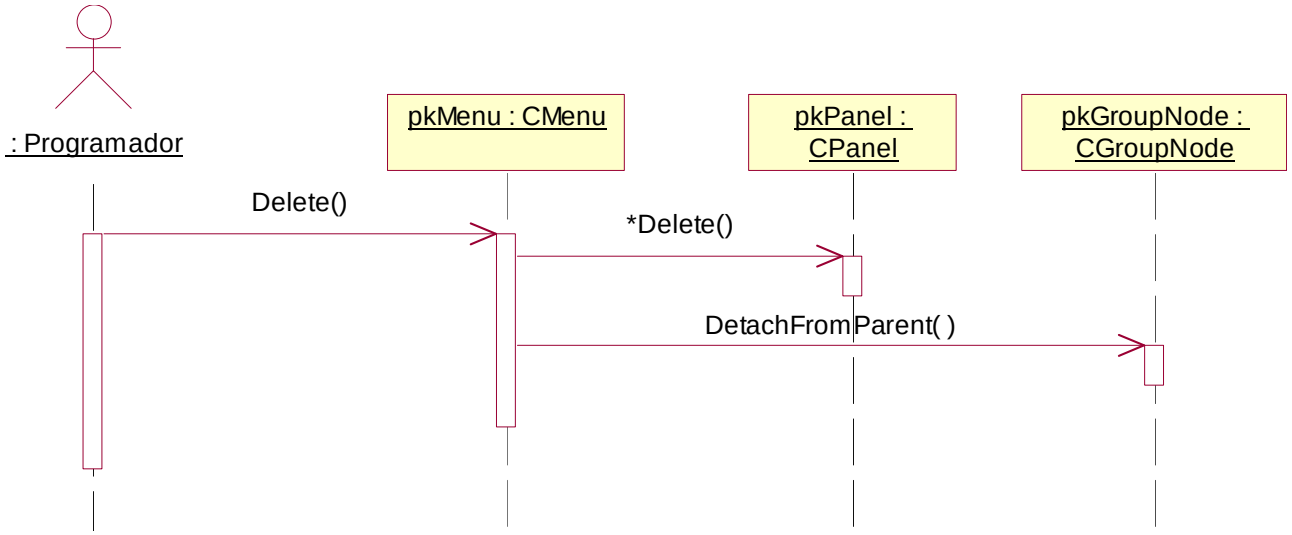

Figura 37 Diagrama de secuencia. "Eliminar menú".

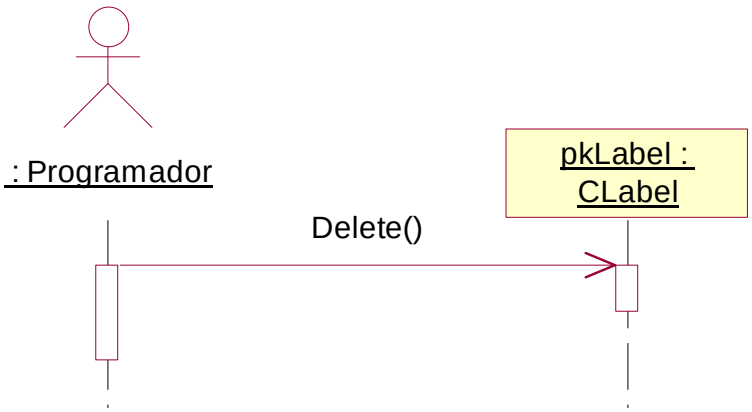

Figura 38 Diagrama de secuencia. "Eliminar etiqueta".

3.1.7 Realización del caso de uso “Salvar o cargar interfaz”

En este paquete, el objeto de tipo *CImageControl* puede sustituirse por objetos de tipo *CButton* y *CPanel* al heredar estos últimos de *CImageControl*.

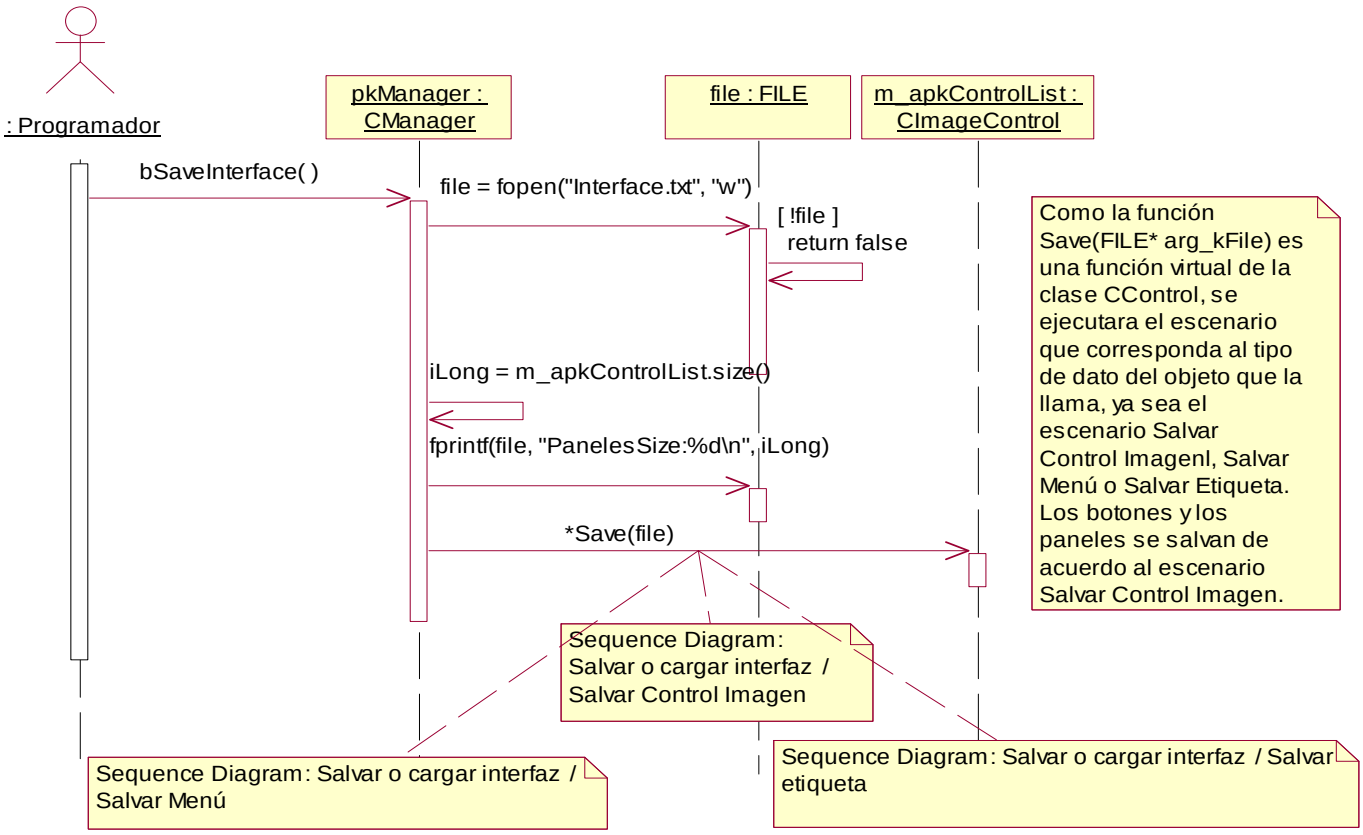

Figura 39 Diagrama de secuencia “Salvar interfaz”.

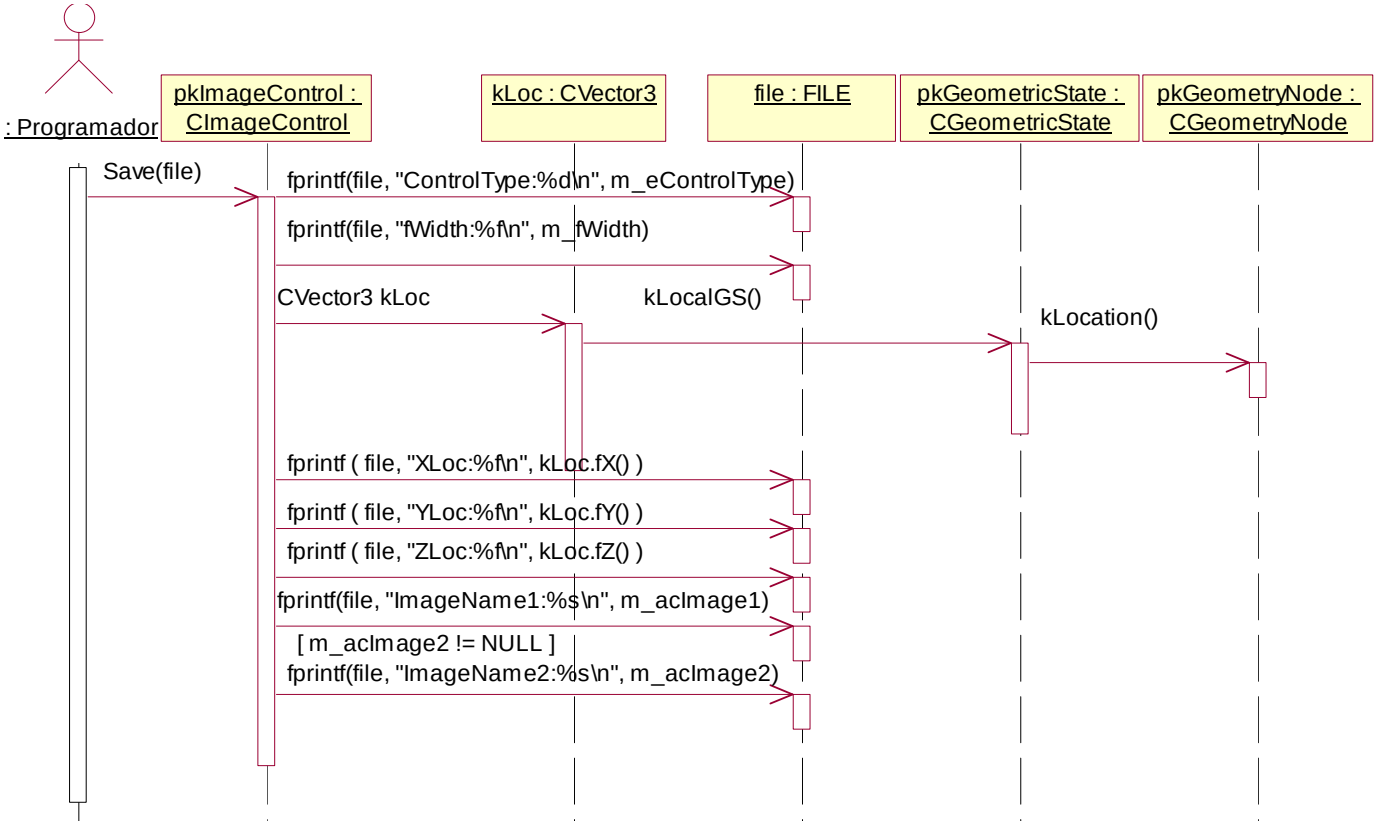

Figura 40 Diagrama de secuencia "Salvar control imagen".

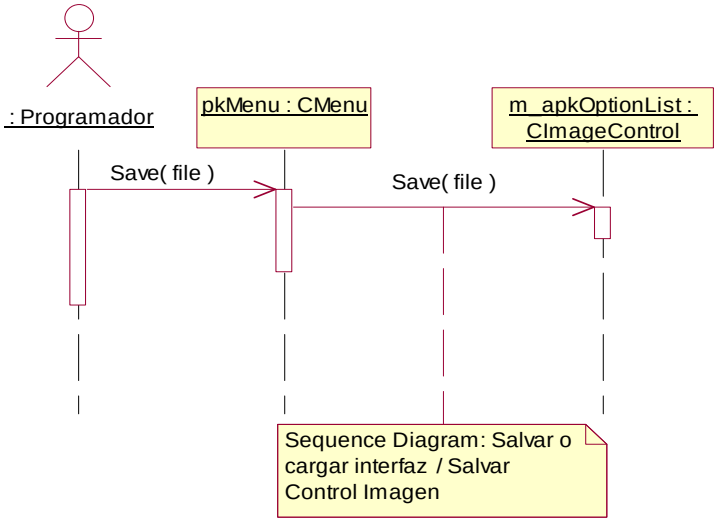

Figura 41 Diagrama de secuencia "Salvar menú".

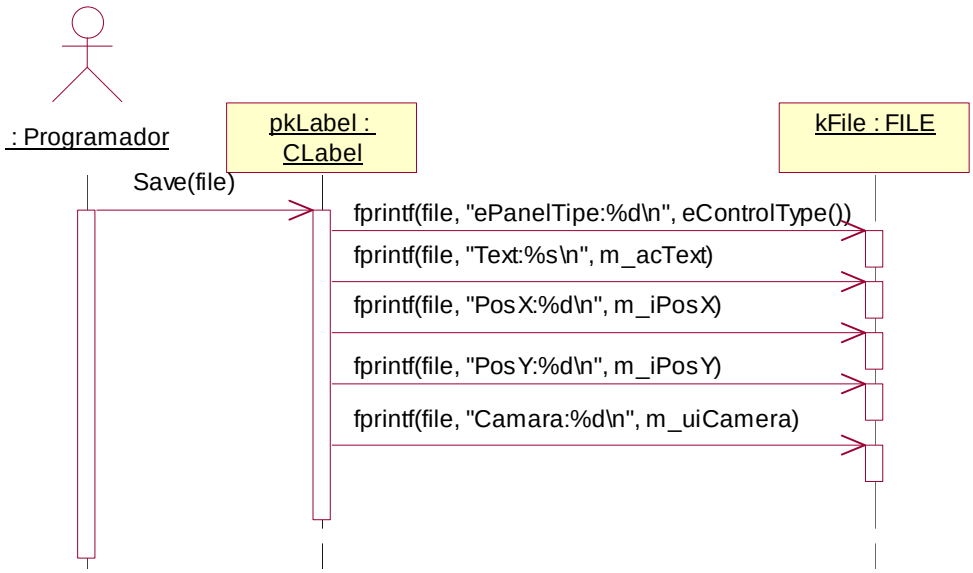

Figura 42 Diagrama de secuencia "Salvar etiqueta".

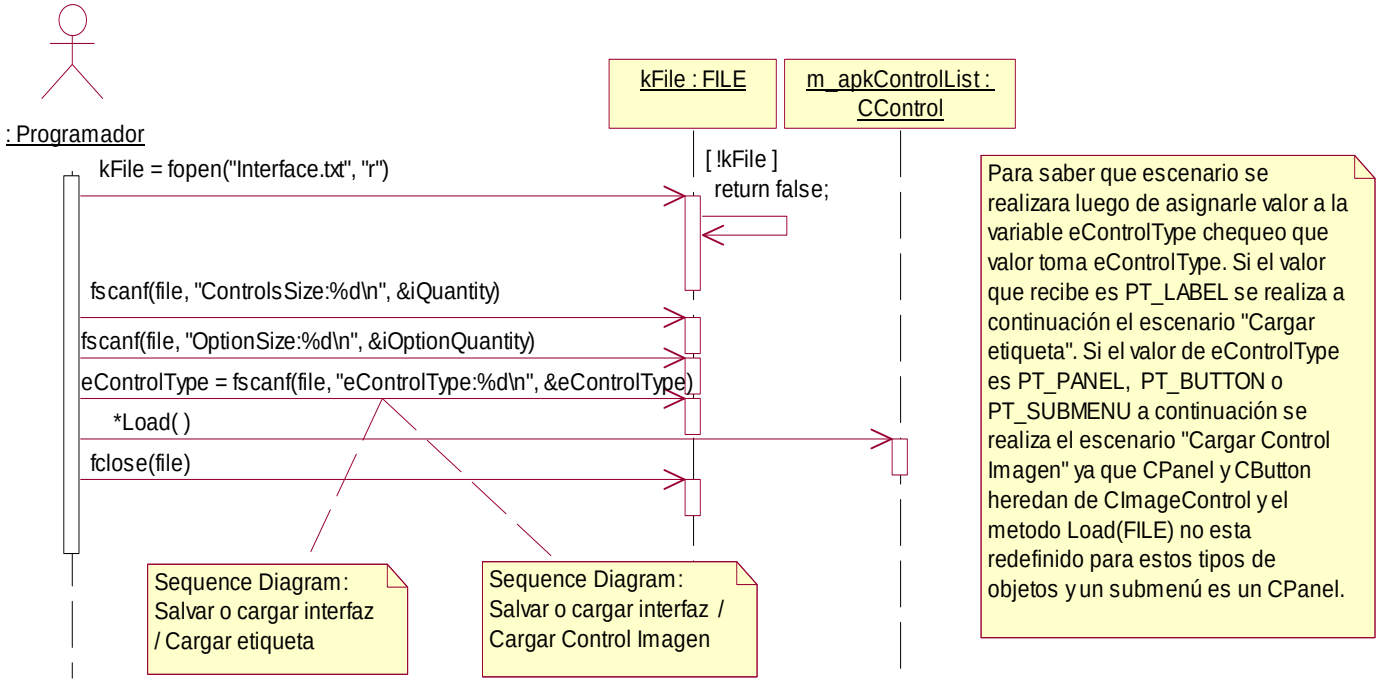

Figura 43 Diagrama de secuencia "Cargar interfaz".

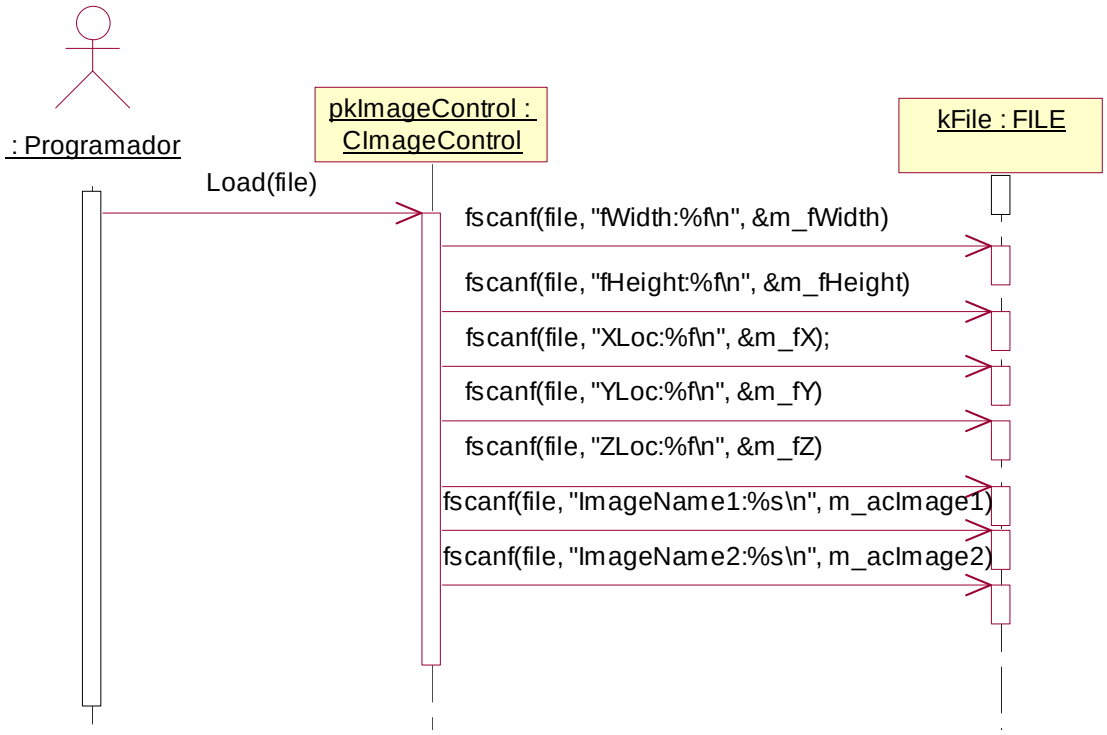

Figura 44 Diagrama de secuencia "Cargar control imagen".

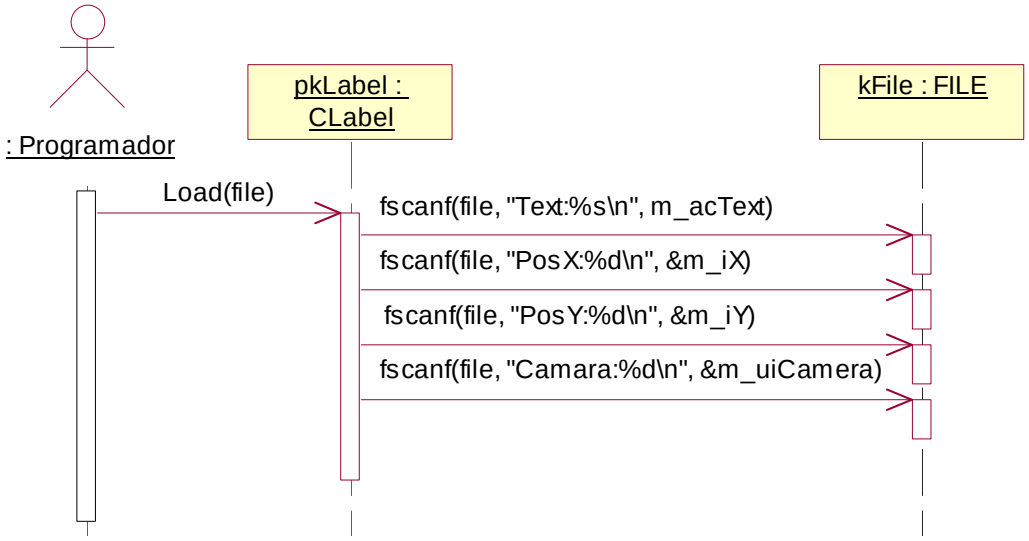

Figura 45 Diagrama de secuencia "Cargar etiqueta".

3.2 Diagramas de clases de diseño.

3.2.1 Diagrama de clases de diseño por paquetes

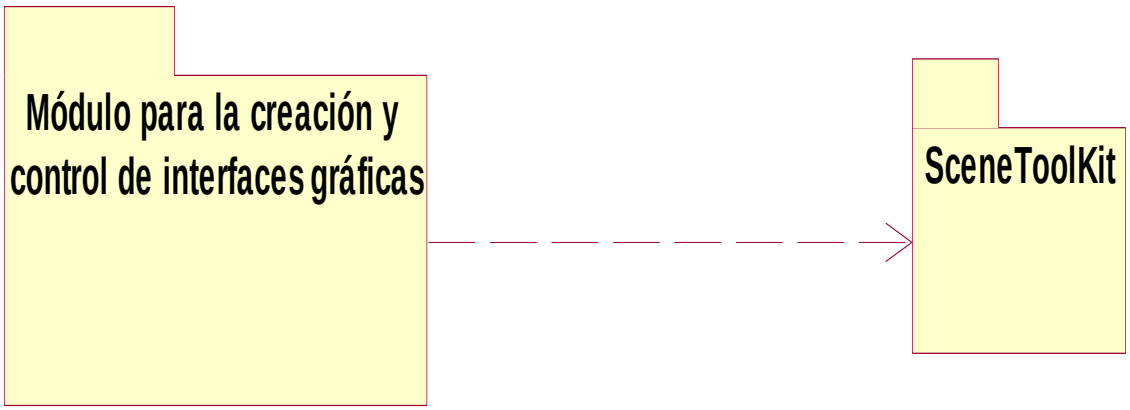

Figura 46 Diagrama de clases de diseño por paquetes.

3.2.2 Paquete "*SceneToolKit*".

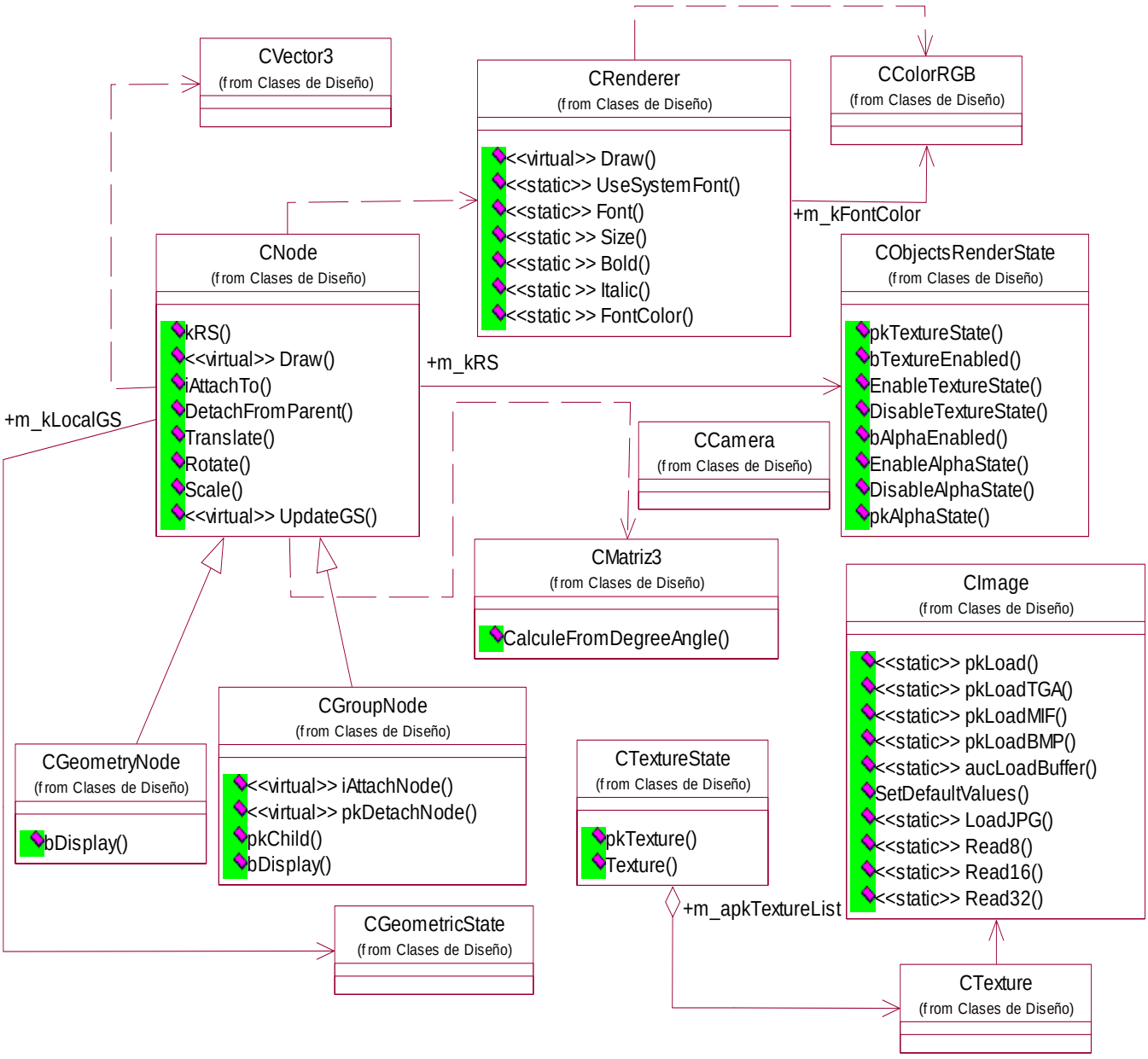

Figura 47 Paquete "*SceneToolKit*".

3.2.3 Paquete “Módulo para la creación y control de interfaces gráficas”

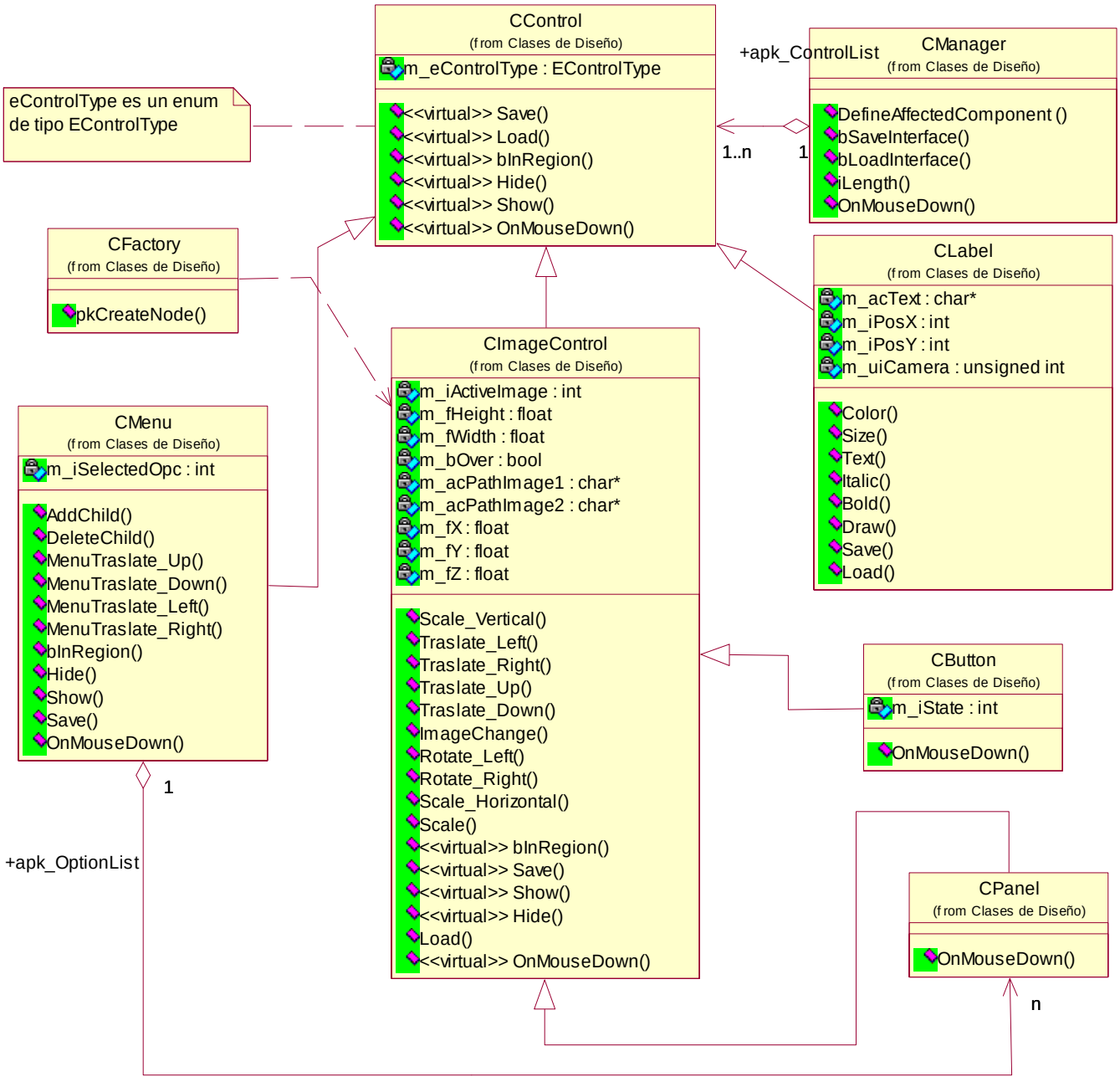

Figura 48 Paquete “Módulo para la creación y control de interfaces gráficas”

3.2.4 Relación entre las clases de los paquetes.

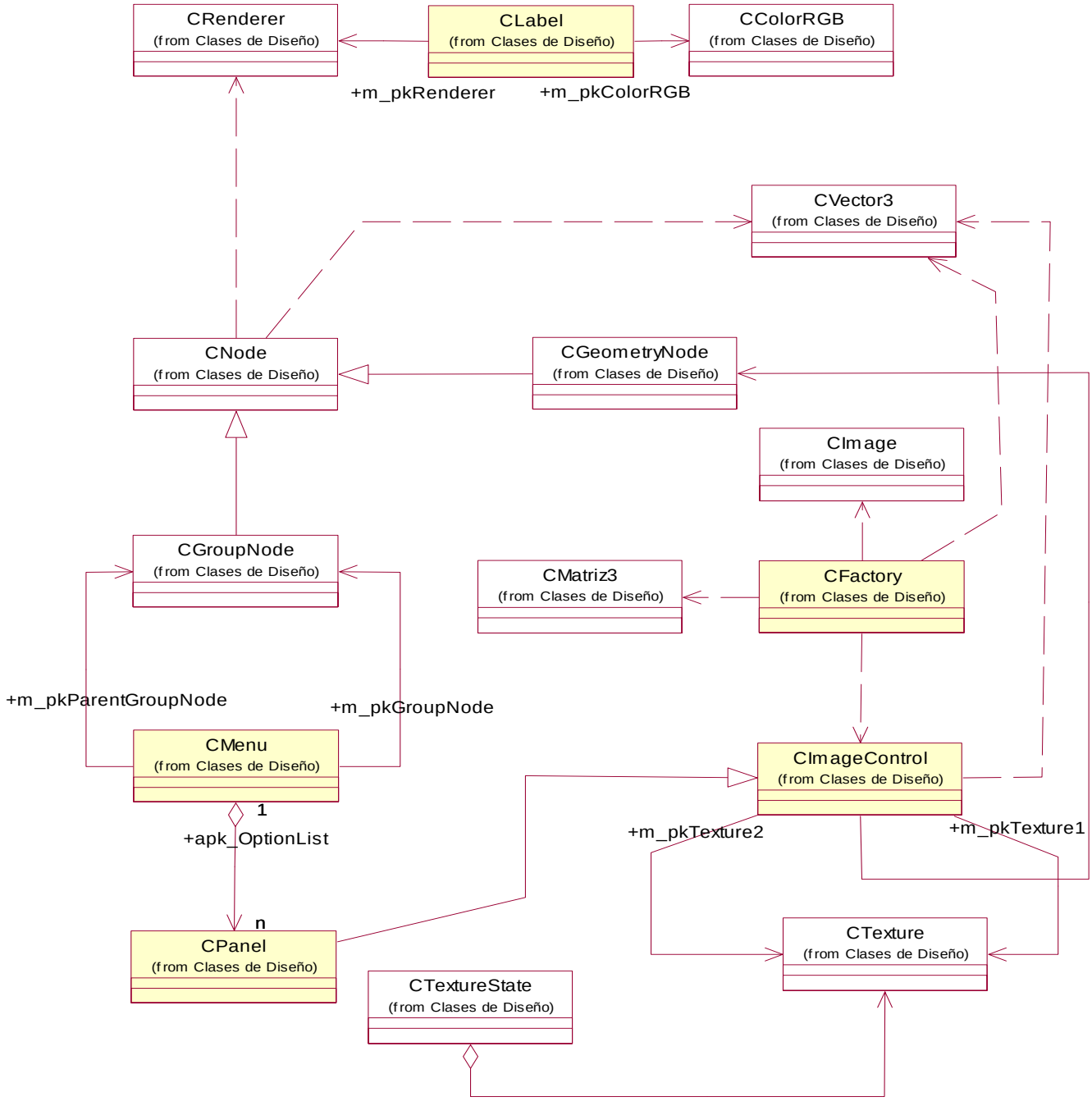

Figura 49 Relación entre las clases de ambos paquetes.

3.3 Descripción de las clases de diseño

Nombre: *CImageControl*	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_iActiveImage	*int*
m_fHeigh	*float*
m_fWhith	*float*
m_bOver	*bool*
m_acPathImage1	*char**
m_acPathImage2	*char**
m_pkTexture1	*CTexture**
m_pkTexture2	*CTexture**
m_pkGeometryNode	*CGeometryNode**
m_fX	*float*
m_fY	*float*
m_fZ	*float*
Para cada responsabilidad:	
Nombre:	*PlanoRotate_Right()*
Descripción:	Rota el control imagen a favor de las manecillas del reloj.
Nombre:	*PlanoRotate_Left()*
Descripción:	Rota el control imagen en contra de las manecillas del reloj.
Nombre:	*PlanoTraslate_Left(float arg_iIncremento)*
Descripción:	Traslada el control imagen en dirección del eje “X” negativo.
Nombre:	*PlanoTraslate_Right(float arg_iIncremento)*
Descripción:	Traslada el control imagen en dirección del eje “X” positivo.
Nombre:	*PlanoTraslate_Up(float arg_iIncremento)*
Descripción:	Traslada el control imagen en dirección del eje “Y” positivo.

Nombre:	*PlanoTraslate_Down(float arg_iIncremento)*
Descripción:	Traslada el control imagen en dirección del eje “Y” negativo.
Nombre:	*PlanoScale(float arg_fMax)*
Descripción:	Escala el control imagen lo mismo vertical que horizontalmente.
Nombre:	*PlanoScale_Vertical(float arg_fMax)*
Descripción:	Escala el control imagen verticalmente.
Nombre:	*PlanoScale_Horizontal(float arg_fMax)*
Descripción:	Escala el control imagen horizontalmente.
Nombre:	*ImageChange()*
Descripción:	Cambia la textura de la maya del control imagen por otra definida para el.
Nombre:	*bInRegion(float arg_fX , float arg_fY)*
Descripción:	Devuelve verdadero o falso si el *Mouse* se encuentra o no sobre el control imagen.
Nombre:	*Save(FILE* arg_kFile)*
Descripción:	Salva los valores de los atributos del control imagen en un fichero de extensión .txt
Nombre:	*Load(FILE* arg_kFile)*
Descripción:	Carga de un fichero de extensión .txt los valores de los atributos del control imagen.
Nombre:	*Show()*
Descripción:	Muestra visualmente el control imagen.
Nombre:	*Hide()*
Descripción:	Oculta visualmente el control imagen.
Nombre:	*OnMouseDown()*
Descripción:	Permite a las clases hijas redefinir este método.

Tabla 20 Descripción de la clase *CImageControl*.

Nombre: *CMenu*	
Tipo de clase: Controladora	
Atributo	**Tipo**

m_iSelectedOpc	*int*
m_apkOptionList	*vector<CPanel*>*
m_pkParentGroupNode	*CGroupNode**
m_pkGroupNode	*CGroupNode**
Para cada responsabilidad:	
Nombre:	*AddChild()*
Descripción:	Adiciona un panel que representa una opción del menú al CGroupNode del menú.
Nombre:	*DeleteChild()*
Descripción:	Elimina un panel que representa una opción del menú del CGroupNode del menú.
Nombre:	*MenuTraslate_Left(float arg_iIncremento)*
Descripción:	Traslada el menú en dirección del eje “X” negativo.
Nombre:	*MenuTraslate_Right(float arg_iIncremento)*
Descripción:	Traslada el menú en dirección del eje “X” positivo.
Nombre:	*MenuTraslate_Up(float arg_iIncremento)*
Descripción:	Traslada el menú en dilección del eje “Y” positivo.
Nombre:	*MenuTraslate_Down(float arg_iIncremento)*
Descripción:	Traslada el menú en dirección del eje “Y” negativo.
Nombre:	bInRegion(float arg_fX , float arg_fY)
Descripción:	Devuelve verdadero o falso si el *Mouse* se encuentra o no sobre el menú y procede a cambiar la imagen de la opción donde se encuentra.
Nombre:	*Save(FILE* arg_kFile)*
Descripción:	Salva los valores de los atributos del menú en un fichero de extensión .txt
Nombre:	*Show()*
Descripción:	Muestra visualmente el menú.
Nombre:	*Hide()*
Descripción:	Oculta visualmente el menú.
Nombre:	*OnMouseDown()*
Descripción:	Responder al evento del *mouse* cambiando la imagen.

Tabla 21 Descripción de la clase *CMenu*.

Nombre: *CControl*	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_eControlType	*EControlType*
Para cada responsabilidad:	
Nombre:	*Save() = 0*
Descripción:	Es un método virtual puro cuyo objetivo es que las clases hijas lo redefinan para salvar el control en cuestión.
Nombre:	*Load() = 0*
Descripción:	Es un método virtual puro cuyo objetivo es que las clases hijas lo redefinan para cargar el control en cuestión.
Nombre:	*OnMouseDown() = 0*
Descripción:	Permitir a las clases hijas redefinir este método.

Tabla 22 Descripción de la clase *CControl*

Nombre: *CPanel*	
Tipo de clase: Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	*OnMouseDown()*
Descripción:	Responder al evento del *mouse* cambiando la imagen si la tiene definida para ello.

Tabla 23 Descripción de la clase *CPanel*

Nombre: *CManager*	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_apkControlList	*vector<CImageControl*>*
Para cada responsabilidad:	
Nombre:	*DefineAffectedComponent(float arg_fX, float arg_fY)*
Descripción:	Devuelve verdadero si se realiza la operación de encontrar sobre que componente se encuentra el *mouse* y actualizar la imagen de ese componente y devuelve falso en caso contrario.
Nombre:	*bSaveInterface()*
Descripción:	Devuelve verdadero si se realiza la operación de salvar todos los valores de los atributos de todos los componentes de la interfaz y devuelve falso en caso contrario.
Nombre:	*bLoadInterface()*
Descripción:	Devuelve verdadero si se realiza la operación de cargar todos los valores de los atributos de los componentes de la interfaz asignándoselos a cada uno de ellos y falso en caso contrario.
Nombre:	*DrawLabels()*
Descripción:	Dibuja en tiempo de ejecución las etiquetas que deben ser visibles.
Nombre:	*iLenght()*
Descripción:	Devuelve la cantidad de componentes de la lista.
Nombre:	*AddControl(CPanel*)*
Descripción:	Adiciona un panel a la lista.
Nombre:	*pkObtainControl(int arg_ipos)*
Descripción	Devuelve el componente que se encuentra en la posición que se le pasa al método.

Tabla 24 Descripción de la clase *CManager*.

Nombre: *CButton*	
Tipo de clase: Controladora	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	*OnMouseDown()*
Descripción:	Cambia la imagen del botón.

Tabla 25 Descripción de la clase *CButton*.

Nombre: *CFactory*	
Tipo de clase: Controladora	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	*pkCreateNode(float arg_fWidth, float arg_fHeight, float arg_XLoc, float arg_YLoc, float arg_ZLoc, CImage* arg_acImagen,bool arg_bTransparent, CGroupNode* pkGroupNode, CPanel* pkPanel, CImage* arg_acImagen2 = NULL, bool arg_bTransparent0 = 0)*
Descripción:	Crea nodos de geometría (*CGeometryNode*) que son asignados a un *CImageControl* y vinculados al nodo de geometría de la escena.

Tabla 26 Descripción de la clase *CFactory*.

Nombre: *CLabel*	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_acText	*char**
m_pkRenderer	*CRenderer**
m_uiCamera	*unsigned int*
m_kColor	*CColorRGB*
Para cada responsabilidad:	
Nombre:	*Color(CColorRGB arg_kColor)*
Descripción:	Cambia el color de la etiqueta por el que se le pasa por parámetro al método.
Nombre:	*Italic(bool arg_bValue)*
Descripción:	Cambia el tipo de letra a cursiva si el argumento que se le pasa por parámetro al método es verdadero y a normal si el argumento que se le pasa por parámetro al método es falso.
Nombre:	*Size(unsigned int arg_uiSize)*
Descripción:	Cambia el tamaño de la letra de la etiqueta al especificado por parámetro en el método.
Nombre:	*Bold(bool arg_bValue)*
Descripción:	Pone las letras de la etiqueta en negritas si el argumento que se le pasa por parámetro al método es verdadero y a normal si el argumento que se le pasa por parámetro al método es falso.
Nombre:	*Save(FILE*)*
Descripción:	Salva los valores de los atributos de la etiqueta a un fichero.
Nombre:	*Load(FILE*)*
Descripción:	Carga los valores de los atributos de la etiqueta de un fichero y se los asigna a la etiqueta.
***Nombre**:*	*Draw()*
Descripción:	Dibuja la etiqueta.

Tabla 27 Descripción de la clase *CLabel.*

3.4 Definiciones de diseño que se aplican.

Los **Patrones de diseño** permiten establecer un vocabulario común de diseño cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia sobre dichos problemas y soluciones. Son también un gran mecanismo de comunicación para transmitir la experiencia de los ingenieros y diseñadores experimentados a los no tan experimentados, convirtiéndose en unas de las vías para la gestión del conocimiento.

Uno de estos patrones es el que se le llama fábrica, factoría o *factory* y no es más que una clase que implemente uno o más métodos de creación, que son los métodos que se encargan de crear instancias de objetos (estas instancias pueden ser de esta misma clase o de otras). Esta clase tiene entre sus responsabilidades la creación de instancias de objetos, pero puede tener también otras responsabilidades adicionales. Los métodos de creación pueden ser estáticos.

Existen diferentes "tipos" de fábricas: *Abstract Factory, Factory Method* y *Simple Factory*. El tipo que se utiliza en este proyecto es el *Simple Factory* en el caso de la clase *Factory* cuya finalidad no es más que la creación de nuevas instancias de objetos *CGeometryNode*.

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema. Ya se puede pasar a la etapa de implementación del proyecto.

Capítulo 4 Implementación.

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .h y .cpp correspondiente a la implementación en C++.

4.1 Diagrama de despliegue.

Este diagrama consta de una sola computadora personal, por lo tanto se considera no es necesario reflejarlo en este proyecto.

4.2 Diagrama de componentes.

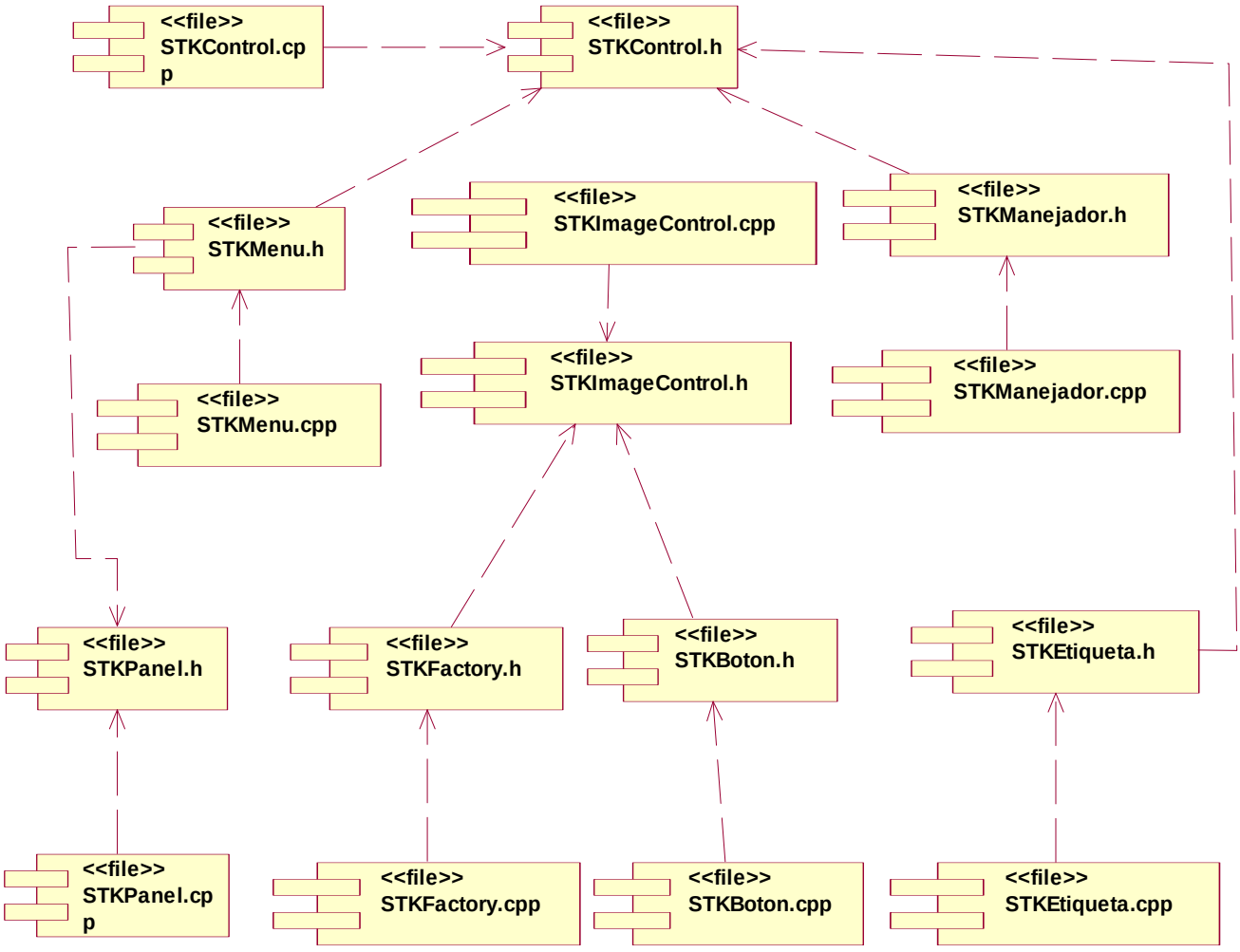

Figura 50 Diagrama de componentes.

Conclusiones

A partir de lo realizado en este capítulo se puede pasar a la etapa de programación de los casos de uso desarrollados para el primer ciclo.

Conclusiones

Para el cumplimiento de los objetivos de este proyecto, y a los requisitos del cliente, se requirió primeramente hacer un estudio las bibliotecas existentes que tratan el tema de la elaboración de componentes visuales. En el estudio se analizaron las características de estos componentes y la estructura más óptima para relacionarlos jerárquicamente. Además, a partir de esa investigación se proponen las características técnicas de la solución.

Posteriormente se hizo la captura de los requisitos funcionales y no funcionales y la agrupación de los primeros en los casos de uso del sistema. Se definió además una primera fase de desarrollo del proyecto. A continuación se transitó por las etapas de análisis, diseño e implementación utilizando los artefactos de RUP, donde surgieron y maduraron las clases, se realizaron los casos de uso a desarrollar en la primera fase y se creó el diagrama de componentes.

Se cumplió con el objetivo de este trabajo que es la implementación de un módulo para la creación y control de interfaces gráficas para sistemas de realidad virtual.

Como resultado obtenido por este trabajo vale destacar que fue probado en la Feria Informática Internacional 2007, en la interfaz del puesto del instructor del Simulador de camión MAZ-700.

Recomendaciones

Se recomienda los siguientes aspectos:

- Elaborar una herramienta visual que facilite el trabajo de los desarrolladores de interfaces visuales.
- Crear un formato propio que permita conservar la integridad de los datos que son salvados.
- Crear una ayuda para futuros desarrolladores.

Referencias bibliográficas

1. CORTIJO, F. J. and F. BERZAL. *Curso de C++ Builder*. [2006] Disponible en: http://elvex.ugr.es/decsai/builder/index.html .
2. *Microsoft Foundation Class Library*. Whatis.com, 2005. [2006]. Disponible en: http://whatis.techtarget.com/definition/0,,sid9_gci214094,00.html
3. *msdn*. Microsoft Corporation[2006]. Disponible en: http://msdn.microsoft.com/library.
4. RIVILLA, F. J. *Programación con VISUAL C++*, [1997]. [Disponible en: http://www.monografias.com/trabajos5/visualcurso/visualcurso.shtml .

Glosario de abreviaturas

SRV: Sistema de realidad virtual

RV: Realidad virtual

UCI: Universidad de las ciencias informáticas.

3D: Tercera dimensión.

API: *Application Programmer's Interface* (interfaces para programadores de aplicaciones).

VCL: *Visual Component Library* (Biblioteca de componentes visuales de la *Borland*)

MFC: *Microsoft Fundation Class* (Biblioteca fundamental de clases de *Microsoft*).

SGI: *Silicon Graphics Incorporated*.

OpenGL: *Open Graphics Library* (Biblioteca de gráficos abierta).

Glosario de términos

Glosario de términos del dominio

Imagen: Un archivo codificado que, al abrirlo, muestra una representación visual de algo (ya sea fotografía, gráfica o dibujo)

Botón: Es una metáfora común utilizada en interfaces gráficas con objetivo símil al de un botón corriente. Los botones suelen ser representados como rectángulos con una leyenda o icono dentro, generalmente con efecto de relieve. Consta de dos imágenes que cambian en dependencia del estado del botón (oprimido o no oprimido). El usuario puede pulsarlo para efectuar acciones indicadas por él.

Menú: Es una imagen o conjunto de imágenes que representan diferentes opciones.

Etiqueta o Label: Muestra texto que el usuario no puede seleccionar ni manipular. Se usa para mostrar textos de título, encabezamientos o incluso para mostrar resultados, ya que puede establecerse su valor (propiedad *Caption*) en tiempo de ejecución.

Glosario de términos general

B:

Bitmap: Los archivos con extensión .bmp, en los sistemas operativos *Windows*, representan la sigla Bitmap, o sea, mapa de bits.

C:

Componente: Elementos genéricos con una funcionalidad muy concreta, cuya única finalidad es la reutilización.

E:

Eventos de un componente: Cualquier suceso que puede ocurrirle a un componente (movimiento del *mouse*, pulsación de algún botón del *mouse*, pulsación de una tecla, desplazamiento o redimensionamiento de una ventana) que pueden condicionar el comportamiento y apariencia del programa.

I:

Interfaz de usuario: Forma en que los usuarios pueden comunicarse con una computadora y comprende todos los puntos de contacto entre el usuario y el equipo.

M:

Módulo: Componente autocontrolado de un sistema que posee una interfaz bien definida hacia otros componentes.

Métodos de un componente: Funciones asociadas al componente que pueden invocarse para que el componente realice distintas acciones.

P:

Productos finales: Sistemas de Realidad Virtual, dígase juegos y simuladores.

Propiedades de un componente: Elementos del componente que configuran su aspecto y controlan su comportamiento.

R:

Realidad Virtual: Es un sistema o interfaz informático que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad, una realidad ilusoria, pues se trata de una realidad perceptiva sin soporte objetivo, sin red extensa, ya que existe sólo dentro del ordenador. Por eso puede afirmarse que la realidad virtual es una pseudorealidad alternativa, perceptivamente hablando.

S:

Sistema de Realidad Virtual: Sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

Simulador: Aparato que permite la simulación de un sistema, reproduciendo su comportamiento. Los simuladores reproducen sensaciones que en realidad no están sucediendo.

T:

Textura*:* Imagen que sirve de “piel” a los modelos en un mundo virtual**.**

Índice de figuras y tablas

Índice de figuras

Índice de tablas