Proyecto de fin de carrera
de Facultad de Ingeniería
de la Universidad de la República
Oriental del Uruguay.

CREADO POR:
Fabián Arocena,
Luis Briosso
y Santiago Schettini.

TUTORES:
Ing. Alvaro Pardo e Ing. Pablo Flores.

MONTEVIDEO, URUGUAY, AGOSTO 2006.

1 Resumen

El objetivo de este documento es explicar detalladamente todo el proceso de elaboración del proyecto llamado Skopeo, desde sus requerimientos iniciales hasta las características finales y las metodologías usadas para desarrollarlo. Además de explicar el trabajo de diseño y de desarrollo del proyecto, se mostrará las diferentes cualidades de las herramientas usadas y el proceso de elección de las mismas; finalmente se anexarán manuales de uso del programa, tanto como usuario o como desarrollador de plugins.

2 Introducción

La importancia de la información digital de audio y video ha crecido de manera vertiginosa en los últimos años. Así mismo, lo han hecho las tecnologías de procesamiento de este tipo de datos, y su rol en nuestra vida cotidiana.

Este proyecto forma parte de la creación de una herramienta de tratamiento y manipulación de información digital audiovisual. Esto implica el hecho de haber integrado el funcionamiento de un proyecto pensado para generar esta herramienta, el VAP (Video Audio Processing).

El VAP es una librería de tratamiento de video y audio escrita en C++ y basada en cuatro librerías multiplataforma de código abierto: FFMPEG, ITK, FOBS y XERCES. Este aporta las herramientas básicas para el desarrollo de algoritmos de procesamiento de audio y video, así como funcionalidades descriptoras de información audiovisual.

Esta librería requería de un sistema que ampliara las posibilidades de interacción con los usuarios.

Para cumplir con esta necesidad, este proyecto estableció como objetivos principales el diseñar e implementar la arquitectura de plugins, efectuar algunos plugins de prueba, desarrollar una interfaz grafica, investigar el estado del arte de los elementos a usar y escribir la documentación necesaria.

Todos estos objetivos serán explicados en más detalle en la siguiente sección de este documento.

Es importante resaltar la independencia en que trabajan los dos proyectos por más que una versión estable del VAP sea parte del funcionamiento de este.

2.1 Objetivos y alcance del proyecto

Una vez establecidas las características deseadas, se delinearon los principales objetivos del proyecto Skopeo. Estos se enumeran a continuación:

- Investigar el estado del arte de las posibles tecnologías a usar.
- Obtener una aplicación en funcionamiento que cuente con las siguientes características:
- Comunicación fluida con los elementos del VAP.
- Reproducción de video utilizando el VAP.
- Interfaz gráfica.
- Posibilidad de extender la aplicación una vez finalizada (mediante la incorporación de plugins).
- Ser multiplataforma

La investigación del estado del arte de todas las herramientas que se utilizarían fue de vital importancia para poder cumplir con los restantes objetivos del proyecto. Se estudiaron varias áreas relacionadas al tema, por ejemplo, el funcionamiento del VAP, qué librerías se podrían usar para desarrollar la interfaz gráfica, etc.

En cuanto al desarrollo de la aplicación se plantearon los siguientes prioridades:

Comunicación con el VAP. Absolutamente necesaria para poder contar con todas las funcionalidades que este brinda.
Reproducción de video. Vital para poder visualizar los resultados de los diferentes algoritmos a implementar.
La interfaz gráfica debía ser amigable y útil para los usuarios en general, sencilla y lo más dúctil posible. En ella se debían encontrar los elementos más comunes de una interfaz gráfica, como pueden ser los menús de archivo, edición, ayuda, etc.
Arquitectura de plugins. De suma importancia, ya que daría la libertad al usuario desarrollador de implementar de manera sencilla algoritmos que utilicen las funciones brindadas por el VAP y las que aportan los demás elementos del sistema, una vez que este compilado y operando.
El sistema debía ser diseñado de manera que todos sus componentes sean multiplataforma, pues la idea principal es que pueda ser utilizado tanto sobre el sistema operativo Windows como en Linux.

La documentación era absolutamente necesaria para que los posibles usuarios pudieran comprender el proceso que se llevó a cabo a lo largo del proyecto y que comprendan el funcionamiento del mismo y sus principales componentes.

En resumen, este proyecto se debía cumplir con los objetivos mencionados, siempre teniendo en cuenta las diferentes condiciones que se pedían para el producto final. Skopeo permite ser utilizado en la forma en la que se presenta al momento de su finalización curricular, esto no quita que existan elementos para agregar y mejorar en el futuro.

2.2 Marco Inicial

Los objetivos y exigencias que tenía el proyecto llevaron a trabajar con las siguientes condiciones, restricciones y posibilidades.

El primer desafío que se iba a enfrentar era el de la investigación del estado del arte y de las diferentes variantes existentes para lo que se quería realizar. Se efectuó una larga investigación al mismo tiempo que se iba diagramando la arquitectura del sistema, siempre condicionada una por la otra. El diseño aportaba posibilidades y nuevas rutas de investigación y a su vez, la investigación acotaba el diseño.

Dentro del estado del arte de las tecnologías relacionadas con este proyecto, los aspectos sobre los que era necesario investigar eran el lenguaje a utilizar, la estructura de plugins a desarrollar, las diferentes herramientas que podrían servir para el desarrollo de la arquitectura planteada, librerías de interfaz gráfica, etc.

Se investigaron las diferentes posibilidades existentes para hacer un sistema que incorporara plugins, tanto en el lenguaje usado por este proyecto como en otros lenguajes. Se encontró que la arquitectura usada en la mayoría de sistemas de plugins era una basada en la carga de librerías dinámicas, usando diferentes puntos de comunicación para posibilitar el uso de los elementos del programa principal en los plugins.

Un desafío fundamental sería el de integrar este trabajo al ya existente VAP, lo cual implicaría un estudio minucioso y exhaustivo de sus características y funcionalidades. Dado el hecho de que era el sistema por el cuál se iba a acceder y manipular los archivos, la manera en que manejara los datos era decisiva a la hora de elegir el sistema de reproducción a usar. Otra posibilidad que se presentaba era la de usar otro sistema separado del VAP para solo reproducir el audio y el video, pero esto resultaba muy engorroso.

En cuanto a la interfaz gráfica se encontraron diferentes librerías y otros sistemas que posibilitaban implementarla cumpliendo con las exigencias antes mencionadas. La cantidad de opciones era grande, y efectuando una clasificación de las mismas basada principalmente en datos brindados por personas que las utilizaron y características técnicas de las mismas, se llegó a tomar la decisión final.

De más esta decir que la falta de experiencia de los integrantes del grupo en lo que respecta al desarrollo de software se sumo a las restricciones que existían, así como también la exigencia de contar con un sistema multiplataforma en su totalidad o el diálogo necesario entre las diferentes posibilidades que brinda el VAP y los demás elementos del proyecto.

2.3 Introducción al sistema Skopeo

Para poder cumplir con los objetivos principales que se plantearon al principio del proyecto, se resolvió diseñar la arquitectura separando el sistema total en los siguientes 4 componentes principales:

- **VAP**: Video & Audio Processing
- **iguPlayer**: Programa Principal
- **IGU**: Interfaz Grafica del Usuario
- **Plugins**: Extensiones cargadas dinámicamente

Estos módulos, además de relacionarse unos con otros, interactúan con agentes externos al sistema, como por ejemplo un Usuario y/o Desarrollador. En la ilustración 1 se muestra a grandes rasgos las interacciones principales entre los módulos y los agentes. En esta ilustración se pueden observar los elementos enlistados anteriormente, se ve la interfaz gráfica (IGU, iguFrame es su clase principal), se puede ver al VAP (en naranja con sus respectivas librerías y el área de datos en magenta al fondo), al programa principal en celeste y a los plugins en amarillo.

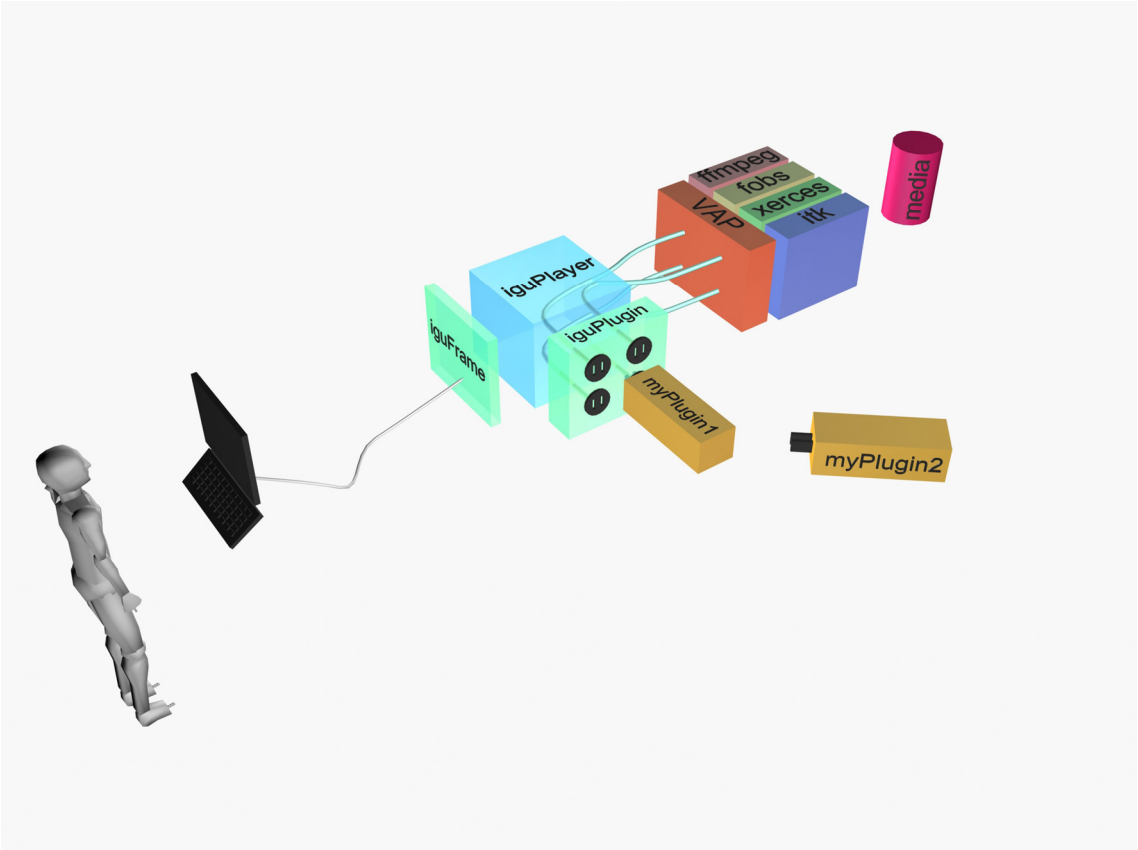


Figura 2-1.- Diagrama general de arquitectura donde se pueden ver los componentes principales de la aplicación, el iguFrame (parte principal de la IGU), iguPlugin (interfaz con plugins), iguPlayer (programa anfitrión) y el VAP con los datos (cilindro magenta en el fondo).

El sistema elegido para que estos módulos y agentes interactúen unos con otros, es el de una conexión mediante interfaces. Dependiendo de qué agente o módulo esté involucrado en qué interacción, actúa tal o cuál interfaz. Cabe destacar que algunos módulos son interfaces en si mismos, como por ejemplo la IGU.

En la figura 2.1, se puede apreciar que la IGU es la que establece la interacción entre el agente Usuario y el VAP. A través de ella el usuario accede la a las funcionalidades de este modulo. Así, el usuario selecciona los archivos a manipular, los procesa, exhibe y almacena. Se ve también la interfaz de la parte de plugins que será explicada más adelante.

2.3.1 VAP

Este sistema consiste en un conjunto de librerías multiplataforma para diversos tipos de procesamiento de audio y video. Dichas librerías implementan diferentes algoritmos de codificación y decodificación audio y video, así como otros de descripción de cuadros, detección de eventos de interés (por Ej. cambios de escena), etc.

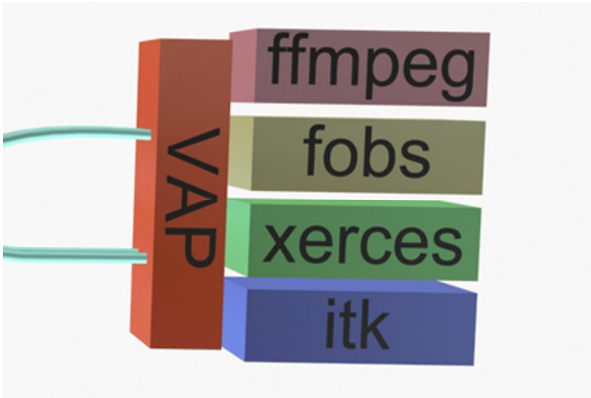


Figura 2-2.- Arquitectura del VAP (Video Audio Processing), se ven las librerías que lo componen, ffmpeg, FOBS, Xerces e ITK.

Tiene como características no funcionales más relevantes:

- facilidad de agregar nuevos algoritmos
- rapidez
- multiplataforma

En la figura 2.2 se presenta un diagrama que describe a grandes rasgos cómo se compone la arquitectura de este módulo. Como se puede apreciar está compuesto por cuatro diferentes módulos que corresponden a librerías que el VAP utiliza para implementar todas las funciones que ofrece. Estas librerías son la FFMPEG (acceso al medio), FOBS (wrapper ffmpeg y C++), XERCES (tratamiento XML) y la ITK (tratamiento video).

Si bien el VAP esta basado a su vez en estos módulos portables que aparecen en la figura 2.2, esta estructura resulta transparente al resto de los componentes del sistema Skopeo.

Nota: Una característica muy importante que posee el VAP es el concepto de Descriptores. Estos contienen información que se extrae de correr algoritmos a frames o shots (grupo de frames) y existe además la posibilidad de almacenarlos en formato XML. Tiene como posibles aplicaciones la detección comerciales, cámaras de vigilancia, etc.

2.3.2 Especificación de Plugins

Los plugins son módulos que se integran con un programa determinado, aportándole funcionalidades no importando que esté compilado. De esto sale la existencia de dos tipos de usuarios para el programa. En primer lugar se encuentra el usuario común, el cual utiliza las funciones presentes en el programa (incluyendo las de los plugins instalados). En segundo lugar se encuentra el desarrollador que para aportar al sistema de una funcionalidad extra, la implementa en forma de plugin programado por él mismo.
Se pueden agregar plugins de manera sencilla, pues la interfaz permite manejar los comandos del mismo programa tanto como generar comandos en el mismo plugin y usarlos sobre un video en el programa principal.

2.3.3 Implementación de Plugins

Se desarrollaron diferentes plugins de tratamiento digital de señales de video para probar el software y para agregar diferentes funcionalidades básicas al sistema. La idea era la de implementar plugins que exijan al programa de forma de probar la performance que se obtuvo.

2.3.4 Interfaz

La interfaz es la parte del programa con la que el usuario interactúa directamente, es lo que ve en la pantalla, son los botones, las barras, los menús, etc.
Por más que un programa sea excelente desde el punto de vista técnico, sino cuenta con el apoyo de una interfaz simple y atractiva, no cumplirá con su función y se hará muy engorroso el utilizarlo, por lo que se intentó que se cumpliera con esas condiciones.

3 Análisis de Requerimientos

3.1 Presentación general

Este proyecto consiste en el desarrollo de una parte de un sistema para edición de audio y video, donde se debe poder extender sus funcionalidades de manera sencilla, mediante la incorporación de plugins. Cuenta además con una interfaz gráfica amigable y está pensada para ser multiplataforma.

3.2 Clientes

Tutores, desarrolladores de plugins y posibles editores de posproducción de audio y video.

3.3 Metas

Las metas principales que se fijaron para este proyecto son las siguientes:

Poder reproducir audio y video offline.
Desarrollar una arquitectura de plugins que permita extender la aplicación.
Desarrollar plugins de prueba.
Crear una interfaz gráfica sencilla de usar.
Editar audio y video offline.

3.4 Funciones

a. Funciones básicas

Ref.	Función	Categoría
R1.1	Reproduce audio y video en tiempo real (play)	Evidente
R1.2	Reproduce video cuadro a cuadro	Evidente
R1.3	Decodifica cierta cantidad de formatos de video	Oculta
R1.4	Adelanta reproducción en curso (forward)	Evidente
R1.5	Retrasa reproducción en curso (rewind)	Evidente
R1.6	Para reproducción en curso (stop)	Evidente
R1.7	Pausa reproducción en curso (pausa)	Evidente
R1.8	Inserta nuevo plugin	Evidente
R1.9	Carga plugin	Evidente
R.10	Ejecuta plugin	Evidente
R.11	Concatena plugins	Evidente
R.12	Modifica parámetros de plugin	Evidente

b. Funciones complementarias

Ref.	Función	Categoría
R2.1	Registra y/o despliega resultados en algún formato de Base de Datos (XML)	Evidente
R2.2	Dibuja sobre video	Evidente

3.5 Atributos del sistema

Atributo	Detalles y restricciones de frontera
Tiempo respuesta	*(Restricción)* Cuando se reproduzca un video no debe de tardar más de una cierto tiempo en comenzar a reproducirlo.
Plataformas	*(Detalle)* Sistemas Linux y Windows
Fácil Manejo	
Interfaz amigable	
Portable	Atributo fundamental del sistema
Adaptable e integrable	Fácil inclusión de plugins

3.6 Casos de uso

Caso de uso 1: Reproducir video en tiempo real (Play). (R1.1)

Actores: Usuario, VAP.
Propósito: Observar el video en pantalla.
Resumen: El usuario abre el programa y en la barra de menú principal selecciona la opción "File". Se despliega un menú, desde el cual el usuario selecciona la opción "Open". Se despliega un cuadro de diálogo de búsqueda de archivos mediante el cual se selecciona el archivo de interés. Luego de seleccionar el archivo, se abre una nueva ventana. A continuación el usuario oprime el botón "Play" de la barra de herramientas principal y el video comienza a reproducirse en dicha ventana.
Tipo: Primario.

Caso de uso 2: Adelantar el video. (R1.4)

Actores: Usuario, VAP
Propósito: Ir hacia delante en la reproducción en curso.
Resumen: El Usuario oprime el botón "Forward" de la barra de herramientas principal, el programa empieza a adelantar el video y el audio hasta que pasa una de dos opciones, o se llega al final del video, o el usuario lo deja de oprimir. En el primer caso, el programa queda mostrando el último cuadro, en el otro caso muestra el cuadro desde el punto en el que se dejó de oprimir "Forward".
Tipo: Primario.

Caso de uso 3: Detener el video. (R1.6)

Actores: Usuario, VAP.
Propósito: Detener la reproducción en curso.
Resumen: El Usuario oprime el botón "Stop" de la barra de herramientas principal y el video que se estaba exhibiendo deja de mostrarse volviendo al estado inicial, esto es si se oprime "Play", comenzara a exhibirse desde el principio.
Tipo: Primario.

Caso de uso 4: Pausar video. (R1.7)

Actores: Usuario, VAP
Propósito: Pausar la reproducción en curso.
Resumen: Mientras se esta exhibiendo un video, el Usuario oprime el botón "Pause" de la barra de herramientas principal. La ventana en donde se estaba desplegando el video queda mostrando el cuadro en el que se encontraba cuando se oprimió "Pause".
Tipo: Primario.

Caso de uso 5: Cargar un nuevo plugin. (R1.9)

Actores: Usuario.
Propósito: Agregar nuevos plugins a la lista de plugins disponibles para su futura utilización.
Resumen: El Usuario abre el programa y selecciona en la barra de Menú principal, la opción "PlugIn". Se despliega un menú del cual selecciona la opción "Load DL". El usuario busca el archivo en cuestión, lo selecciona y el sistema inserta el nuevo plugin en la lista de plugins.
Tipo: Primario.

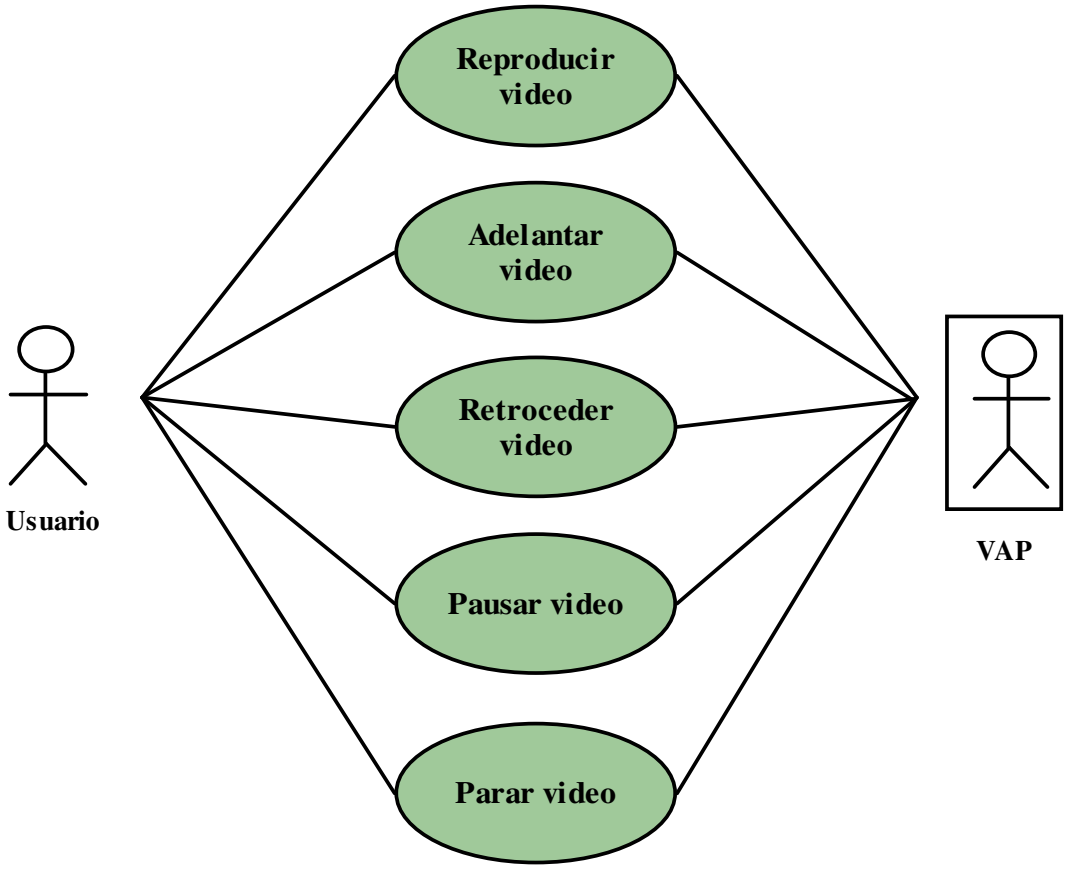
Caso de uso 6: Ejecutar un plugin. (R1.10)

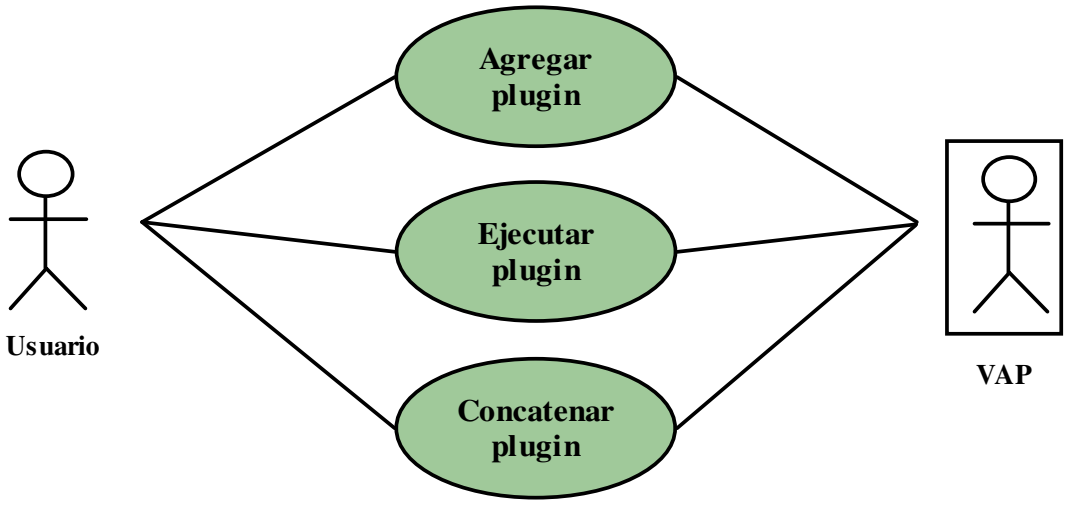
Actores: Usuario, VAP.
Propósito: Usar un plugin para ver alguna propiedad del audio o video.
Resumen: Una vez abierto un video, y cargados los plugins de interés, el Usuario selecciona el plugin que desea de la lista de plugins disponibles. Oprime el botón "Execute" disponible en la barra de herramientas de plugins. A continuación se abre una nueva pantalla que exhibe el resultado de dicho plugin.
Tipo: Primario.

Caso de uso 7: Concatenar dos plugins.(R1.11)

Actores: Usuario, VAP.
Propósito: Observar el resultado de ejecutar dos plugins (en serie).
Resumen: Una vez abierto un video, y cargados los plugins de interés, el Usuario selecciona de la lista de plugins disponibles los 2 plugins que desea concatenar. Oprime el botón "Concatenate" disponible en la barra de herramientas de plugins. Luego, selecciona ejecutar y se abre una nueva pantalla que exhibe el resultado de la aplicación de dichos plugins (en serie).
Tipo: Primario.

3.7 Diagrama de Casos de Uso





4 Selección de herramientas

Luego de conocidos los requerimientos principales del proyecto que se iba a llevar a cabo se plantea la necesidad de buscar las herramientas necesarias para poder implementar en la práctica las soluciones pensadas para el problema. Para ello se realizó una exhaustiva investigación de manera de conocer el estado del arte de los elementos que se necesitaban.

Primero que nada surgió la pregunta más básica, ¿en qué lenguaje de programación se va a desarrollar el programa?. Esta pregunta abre caminos muy diferentes a tomar en cuenta, C++, Java, Phyton, etc. La decisión resulta clave en la evolución del proyecto, pues cualquiera que fuere la opción elegida, la misma condiciona a todas las otras decisiones que se vayan a tomar luego.

Pasada ya la decisión acerca del lenguaje, se planteó la disyuntiva de si se iba a desarrollar todo de cero o si se iba a utilizar alguna librería para la interfaz, por ejemplo. Este detalle hizo que se tuviera que investigar y pasar muchas horas revisando todas las diferentes posibilidades que existen para desarrollar la interfaz gráfica, llegando a probar algunas de ellas para tomar la decisión final.
Luego de tomada la decisión sobre el desarrollo de la interfaz gráfica se planteó el problema de la arquitectura de los plugins, ya pensada la solución desde el punto de vista teórico, se debió buscar una herramienta que nos permitiera implementarla.

Resumiendo, las principales herramientas que se utilizaron para la elaboración del programa son tres, el lenguaje seleccionado (junto a los compiladores usados), la librería usada para la implementación de la interfaz gráfica y por último la librería utilizada para la implementación de la arquitectura de plugins.

4.1 Selección del lenguaje de programación

Las posibilidades para la selección de un lenguaje de programación son muy grandes, en este caso se tiene el trabajo con otro grupo desarrollador como condicionante principal, dado que el grupo que trabaja en el VAP ya venía haciéndolo antes que se empezara con este proyecto, se tuvieron que tomar en cuenta las condiciones en la que estaban programándolo, tanto en cuanto a lenguaje como a compilador a usar.
Dado que el grupo del VAP estaba programando en C++ y sobre el Visual C++, se decidió comenzar a investigar qué tipo de posibilidades existen para el desarrollo de la interfaz gráfica se tenían en ese lenguaje, y en otros que ya eran familiares a los integrantes de nuestro equipo (es importante recordar que los integrantes de este grupo no tenían conocimiento de C++, no así de Java o Python).

Se optó por lo que entendimos era la mejor configuración posible de estas variables: conocimientos de los integrantes sobre desarrollo de software, capacidad del lenguaje para la comunicación con lo desarrollado por el grupo del VAP y diferentes posibilidades que existan para desarrollar la interfaz gráfica y la arquitectura de plugins; se seleccionaron dos lenguajes a tomar en cuenta, Java y C++. Para tomar esta decisión fue importante que los tres integrantes del proyecto conocieran Java y que evidentemente no se podía dejar por fuera a C++ siendo este el lenguaje motor del proyecto.

Hacer una comparación de dos lenguajes como el C++ y el Java no es muy sencillo, dada la complejidad de los mismos y la falta de experiencia que se tenía al principio del proyecto. Por lo tanto se hizo una investigación de qué tipo de características tienen los dos lenguajes en común y cuales eran sus diferencias en los temas que al proyecto más le incumben. Entre estos se encuentran librerías de desarrollo de interfaces gráficas (o en caso de hacerlo todo de cero, qué tan complicado sería y qué beneficios traería con respecto a el uso de librerías), portabilidad en sistemas Windows y Linux, posibilidad de integración (en especial con Java, dado que iba a tener que trabajar con una base en C++), etc.

De cualquier manera, gran parte de lo que se encuentra es muy subjetivo y manejado a favor o en contra de uno de los dos lenguajes, por lo que siempre tuvimos en cuenta el peso importante que tenía en la decisión el hecho que el VAP estuviera escrito en C++.
La tabla T5-1 muestra las diferentes características que tomamos en cuenta a la hora de la decisión final.

Lenguaje	C++	Java
Conocimiento de los estudiantes al comienzo del proyecto	Ninguno	Conocimiento promedio
Utilización del lenguaje en herramientas que el proyecto debe usar	Si, VAP	No
Existencia de librerías multiplataforma para desarrollo de interfaz gráfica	Si	Si
Velocidad del lenguaje en runtime	Buena	No tan buena
Posibilidad de manejo de memoria directamente con puntero a memoria	Si	No

Tabla 4-1.- Diferentes características que se tuvieron en cuenta a la hora de tomar la decisión del lenguaje.

Por más que en ciertos aspectos (en especial, portabilidad) Java resulta mas atractivo, el peso en la decisión que tuvo el hecho de que VAP fuera en C++ y el haber encontrado muchas librerías (y en especial wxWidgets que pareció la más completa y amigable) para trabajar con interfaces gráficas, inclinó la decisión a favor de este último lenguaje.

Ya que se iba a trabajar con el mismo lenguaje que el otro grupo, se decidió usar los mismos compiladores para facilitar el acoplamiento de los proyectos.

4.2 Selección de opción de desarrollo de la interfaz gráfica

La respuesta a cómo hacer la interfaz gráfica puede ser planteada de diferentes maneras. Lo primero que se puede argumentar es que se haga todo de cero, o sea, generar todos los elementos necesarios para poder efectuar todo lo que hace una interfaz gráfica, lo cual resultaba en extremo difícil para el grupo dada la poca experiencia y el poco conocimiento del lenguaje que se tenía al comenzar el proyecto. La otra opción era buscar una librería que abasteciera con diferentes clases, funciones, etc., que pudieran ser usadas para el desarrollo de la interfaz, esta es la opción que se tomó por lo ya mencionado. Lo que quedaba por decidir era qué librería se iba a usar, por lo que se empezó a buscar una librería que cumpliera con los requisitos de que fuera multiplataforma, completa, estéticamente amigable (esto quiere decir que fuera linda y al mismo tiempo flexible), bien documentada, de licencia de código abierto, etc.

Luego de examinar varias opciones algunas de las cuales se fueron descartando, llegamos a cuatro toolkits que evaluamos a continuación. Estos cumplen los requisitos básicos, son C++, son multiplataforma y son de licencia de código abierto.

Las ventajas y desventajas de cada librería listadas en este análisis, fueron obtenidas de las experiencias de múltiples desarrolladores (muchas veces, de las mismas librerías) que comparaban sus librerías con las existentes.

4.2.1 FLTK toolkit

4.2.1.1 Pros:

- La ventaja principal de FLTK es en su tamaño y velocidad. La clase base abstracta para los widgets es de 60 bytes y un “Hello World” linkeado estáticamente es de 80kb aprox.
- Los desarrolladores concuerdan que el código esta cuidadosamente organizado y ofrece al linker muchas oportunidades de dejar de lado código sin usar. El tipo básico de widgets tiene muchas variantes algunas de las cuales son muy innovadoras y hermosas.
- El toolkit es en términos generales muy amigable para principiantes, de fácil entendimiento y la programación es muy directa.
- Contiene documentación comprensible, muchos tutoriales y. LPGL
- Licencia: GNU LPGL
- La portabilidad es muy buena, soportando las siguientes plataformas: Linux, Win32, MacOS

4.2.1.2 Contras:

- La instalación en win32 presenta algunas dificultades al usar Visual C++.
- Nosotros al trabajar con MinGW y DevC++ no encontramos dificultades en absoluto.
- FLTK posee un RAD (Fluid) que no es estable bajo Windows
- Los widgets en Windows no son estándar, y el conjunto de widgets en si es bastante limitado (por ejemplo, no contiene un selector estándar de archivos).
- No soporta propiedades avanzadas de Interfaz Gráfica tal como arrastrar y soltar (drag drop) y clipboard.
- La apariencia de los widgets es bastante básica, sobretodo en Windows, y la customización de los mismos no es amigable.

4.2.2 FOX toolkit

4.2.2.1 Pros:

- FOX es una librería bastante más sofisticada y avanzada que FLTK.
- Es fácil de programar y extender, contiene widgets de alto nivel de buena calidad (como por ejemplo, ventanas separables, listas en ramas, listas de archivos notebooks, etc.)
- Múltiple documentación de soporte de la interfaz
- Apariencia nativa de Windows (bueno o malo, dependiendo del punto de vista)
- Incluye varios controles, arrastrar y soltar y controles de OpenGL.
- Implementa iconos, imágenes, ayuda de status de line y tooltips.
- La portabilidad es uno de los objetivos principales de FOX, y soporta las siguientes plataformas:
    - Win32 (9x/NT/2k/XP): Con los siguientes compiladores: Visual C++ 6.0 & 7.0 ; Borland C++ 5.5 ; Borland C++ ; Cygwin 1.1; MinGW 1.1 ;Open Watcom C++ ; Digital Mars C++ ; IBM Visual Age C++
    - Linux (x86) Compilado con: GNU GCC (2.96 - 3.3)
    - Linux (ia64) " Intel(R) C++ Itanium(R) Compiler 7.1
    - Linux (AMD64) Compilado con GNU GCC (3.3.1) y con GNU GCC 2.96
    - FreeBSD, OpenBSD y NetBSD con el compilador: GNU GCC

4.2.2.2 Contras:

- Utiliza macros en los API´s
- No posee en la actualidad un RAD para desarrollar la interfaz rápidamente y esto es la causa principal de porque nosotros la descartamos.
- Licencia: LPGL

4.2.3 wxWidgets

4.2.3.1 Pros:

- Las aplicaciones utilizando esta librería adoptaran la apariencia nativa cualquiera sea la plataforma en la que se este utilizando.
- La documentación es extensa y detallada y, en contraste con la de FOX, es organizada y amplia. El paquete de distribución contiene un gran número de ejemplos, desarrollados y soportados por la enorme comunidad de wxWidgets.
- Solo el 10% de sus funciones son no portables, las cuales comprenden funciones específicas de dibujo para los sistemas OS, sin embargo, las demás gozan del la propiedad "escrito una vez, corre donde sea"
- La ventaja mas atractiva para nuestro proyecto es que hay un numero considerable de RAD´s, entre los cuales se encuentra un plugin para el Dev C++ el cual es amigable y, aunque contiene algunos bugs, ahorra la programación pesada de los widgets dejando para nosotros la programación de las funciones de interés.
- Soporta archivos XML seguros, e integra OpenGL, HTML y Networking.
- Los compiladores son los siguientes:
    - Visual C++ 1.5, 4.0, 5.0, 6.0, 7.0, 7.1
    - Borland C++ 4.5, 5.0, 5.5
    - Borland C++Builder 1.0, 3.0, X
    - Watcom C++ 10.6 (Win32), OpenWatcom 1.0
    - Cygwin (using configure)
    - Mingw32
    - MetroWerks CodeWarrior (many versions)
    - Digital Mars 8.34+
- Licencia: GNU GPL compatible.

4.2.3.2 Contras:

- Esta librería presenta problemas con el Fedora Core 2 y la desventaja principal es que dada la extensión de la librería, se producen ejecutables bastante pesados. (Un simple editor de texto pesa 3, 4 Megabytes)

4.2.4 GTK+ con gtkmm

- GTK+ es una librería en C desarrollada para el programa GIMP de procesamiento de gráficos 2D, y ha ido evolucionando en un gran proyecto de colaboración.
- Gtkmm es un envoltorio en C++ para GTK+ que reacciona rápidamente a los cambios en el código C de GTK+, y ofrece a los desarrolladores una buena API.
- Básicamente Gtkmm es una extensa librería con buena documentación y una comunidad de soporte enorme, pero la portabilidad no es su característica mas fuerte, y eso por eso que no haremos un análisis muy extenso de esta librería.
- Si bien en sistemas operativos UNIX posee una gran portabilidad, actualmente en Windows no hay versiones de esta librería que sean estables.

4.3 Herramientas de arquitectura de Plugins

A la hora de implementar la arquitectura de plugins seleccionada se tuvo que decidir qué herramientas se iban a usar para que se pudieran acoplar dinámicamente.
La técnica mas utilizada para esta finalidad en el desarrollo de software en C/C++ es la de las librerías dinámicas. Dependiendo del sistema operativo, se utilizan Dynamic Linking Libraries (.dll´s, para Windows) o Shared Objects (.so para sistemas basados en Linux)
Se encontraron diferentes alternativas. Primero que nada se estudió la posibilidad de hacerlo con los paquetes de librerías estándar de C++, pero esto fue descartado luego de varios intentos sin éxito. Luego se pensó en paquetes de librerías que estuvieran diseñados específicamente para la carga de las librerías dinámicas. Esto tenía el gran problema de tener que integrar otro paquete más, hecho que tomaría un tiempo importante. Por lo tanto se resolvió revisar las opciones y herramientas que tenía wxWidgets en lo que respecta al manejo de librerías dinámicas, encontrando que cuenta con varias herramientas que fueron muy útiles a la hora de implementar la arquitectura diseñada.
Las herramientas que wxWidgets aportó para esta parte fueron, primero que nada la clase wxDynamicLibrary que sirve para poder cargar y manejar librerías dinámicas, y siempre tanto en Windows como en Linux; y la clase wxArray, que brinda la posibilidad de manejar un array dinámico de librerías, con esto se puede implementar el manejo de una lista de plugins a ejecutar en el programa.

5 Solución Propuesta

En esta sección se efectuará un recorrido por el diseño propuesto como solución y el camino que se tomó para llegar a él. Luego se efectuará una descripción de las herramientas usadas en la implementación de la solución propuesta. Para finalizar con la sección, se mostrará cómo se utilizaron las herramientas seleccionadas en la solución final.

5.1 Diseño y Arquitectura

En la introducción se explicó de manera sucinta el esquema general del sistema diseñado. En los próximos párrafos describiremos de manera detallada el diseño del sistema en general.

El diseño debía estar orientado hacia las tres cuestiones fundamentales enumeradas anteriormente:

- Poder interactuar con el VAP.
- Fácil integración de plugins .
- Comunicación con el usuario a través de la interfaz grafica.

El método de trabajo de este equipo fue el de encarar un problema a la vez sin perder de vista lo genérico.

El primer problema planteado fue la comunicación con el VAP. Se tenía por un lado todos los elementos a desarrollar como por ejemplo, la interfaz grafica, la arquitectura de plugins, etc. y por otro lado el VAP y los datos. Aquí se debía decidir cómo se iba a lograr acceder a todas las funcionalidades de este componente.

Primero que nada se pensó en una comunicación basada en XML, en donde los datos fueran enviados y recibidos mediante un protocolo especifico y manipulándolos luego desde un módulo que actuara de interlocutor entre ambos componentes. Esta opción permitiría por ejemplo, una comunicación remota similar a la del web service.

La segunda opción, que fue la adoptada al final, fue la de que todo el sistema estuviera ligado directamente al VAP. Esto quiere decir que ambos hacen referencia a los mismos lugares de memoria en disco (Figura 5.1).

Este camino fue elegido por varias razones. La comunicación a través de archivos XML contenía características atractivas; es un lenguaje adecuado para el diseño de un protocolo de comunicación, brinda la posibilidad de realizar la transmisión de información de una manera estructurada y concreta, y además independiente del lenguaje que utilizaran los interlocutores. Sin embargo, implicaba una inversión adicional de tiempo para investigar esta disciplina, y presentaba ciertas dificultades como por ejemplo la generación, transmisión y parseo de los archivos. A su vez, el hecho de que el sistema pudiera comunicarse con el VAP remotamente no era una de las prioridades de este proyecto.

Si bien el interés y las motivaciones de este equipo se inclinaban hacia ese tipo de investigación, por lo antes mencionado se decidió dejar de lado esta opción. Esto se debió a que, utilizando una memoria compartida, el tiempo invertido en aprender C++ pudo ser aprovechado en forma más eficiente y dada la decisión de utilizar este lenguaje, la comunicación entre los módulos fue fluida.
Otro punto a favor de la decisión tomada, fue que a medida que se estudiaba el VAP se mostraba como un sistema amigable al acoplamiento de otros componentes.

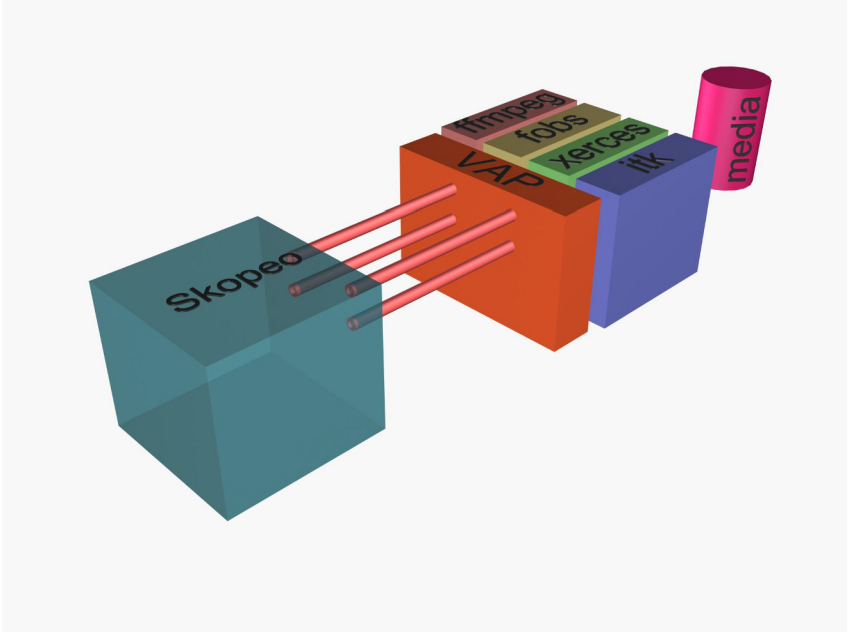


Figura 5-1.- Comunicación con el VAP.

En la figura 5.1 se intenta representar gráficamente el concepto detrás de la comunicación entre el Skopeo y el VAP. Los tubos que los unen simbolizan un ensamblaje entre los dos componentes, lo que posibilita a Skopeo un acceso a las clases y funciones del VAP. En cuanto a la perseverancia, ambos comparten el mismo espacio de memoria.

El siguiente problema abordado fue el de la comunicación con el usuario. Se debía definir la estructura a usar para lograr que los usuarios que quisieran utilizar las diferentes posibilidades de los demás componentes del sistema lo hicieran de una manera intuitiva y práctica.

Se pensó en una arquitectura basada en una clase que maneje las ventanas que se utilizarán, y que contenga todas las herramientas que se mostrarán, tales como el play, el stop, rewind, etc. Además debía abrir y llamar una ventana en la cuál se reproduciría el video (ver Figura 5.2).

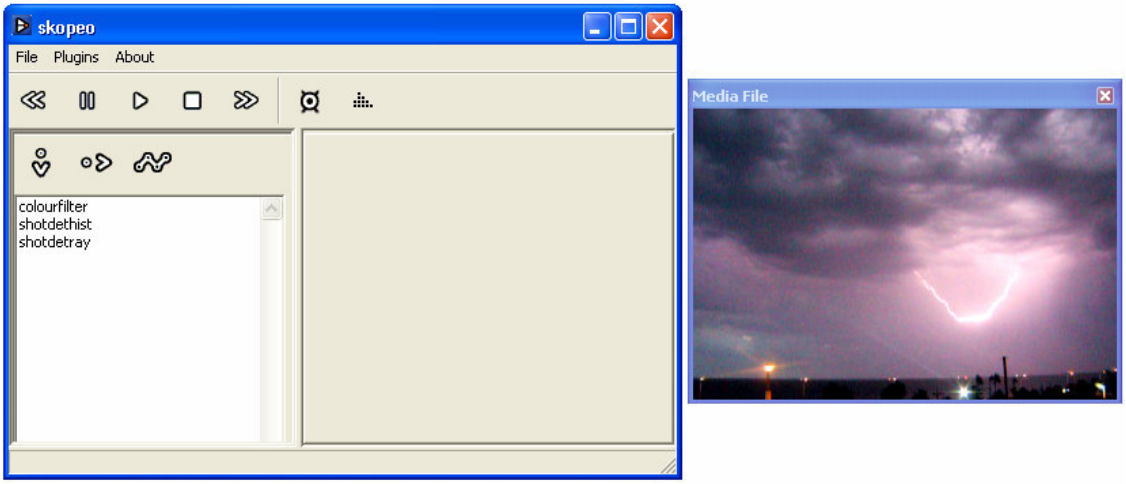


Figura 5-2.- Interfaz gráfica y visor de video.

Se pensó el sistema de manera que se pueda manejar todo desde una misma ventana y que se reproduzera el video en una ventana separada. La idea es que se maneje todo directamente desde la ventana principal, se abran los archivos, se manejen los videos y el audio, se maneje mismo la carga de los plugins, se manejen las diferentes herramientas del VAP, etc.

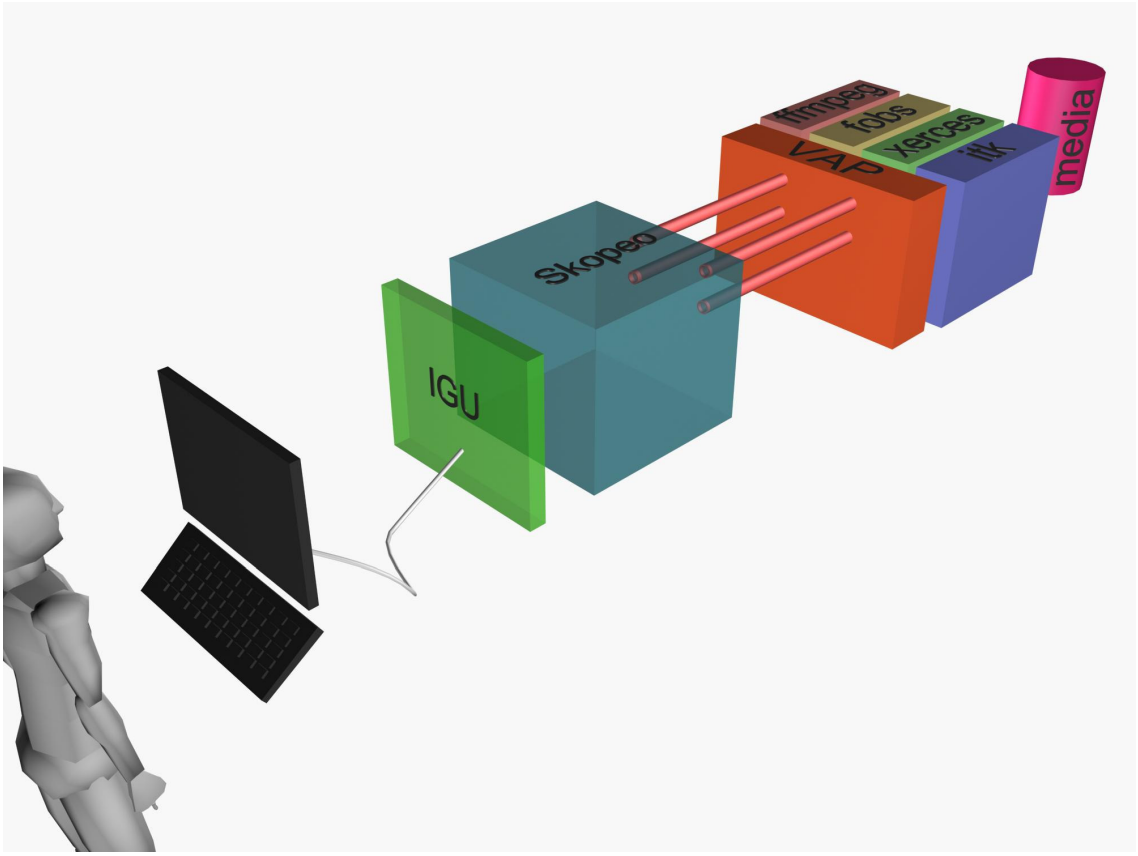


Figura 5-3.- Diseño del proyecto con comunicación con el VAP e interfaz gráfica (GUI).

En la figura 5.3, podemos apreciar la ventana principal, con el visor de video a la derecha. Dicho visor posee propiedades drag & drop y puede ser cerrado independientemente del resto de la aplicación.

El tercer tema a trabajar era el de los plugins. Este era de vital importancia, y el diseño que se llevaría a cabo debía ser simple y al mismo tiempo debía contar con la capacidad de ofrecerle a los usuarios la posibilidad de trabajar con flexibilidad sobre los archivos.
El diseño seleccionado se basó en definir una interfaz que permita a los plugins comunicarse con los demás componentes del programa. Se pensó en una interfaz en la cuál se definan los diferentes elementos que se utilizarían para la comunicación.
Otro elemento importante de la arquitectura de plugins es el módulo que los manejará dentro del programa principal, este controlará una lista de plugins que se cargarán y va a inicializar, ejecutar y destruir todos los elementos necesarios que se precisen a la hora de cargar y usar un plugin.
En resumen, se tienen dos elementos fundamentales en la arquitectura de plugins, una interfaz que define todos los elementos de la comunicación entre los plugins y los demás módulos del programa, y un “manipulador” de plugins que se encarga de administrar una lista de plugins y de cargar, ejecutar y borrar los plugins.

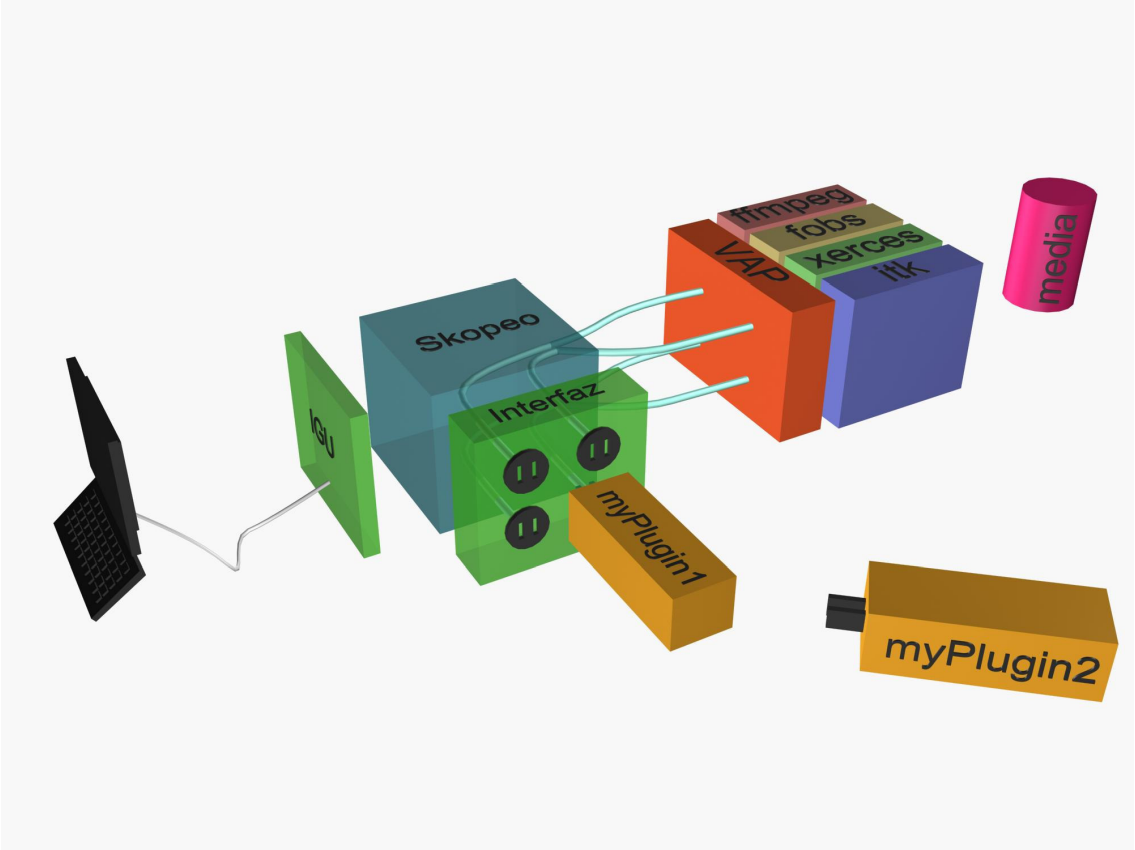


Figura 5-4.- Vista completa del proyecto.

Al finalizar el diseño de la arquitectura del proyecto se unieron todas las piezas para llegar a la arquitectura general que se ve en la figura 4.3. En esta figura se puede apreciar cómo los plugins se conectan al programa principal (Skopeo, luego llamado iguPlayer) y se comunican con el VAP a través del mismo. También se ve que la comunicación del proyecto con el usuario se hace a través de una interfaz, esta interfaz tiene como componente principal la clase contenedora de la ventana principal que se mencionó anteriormente. Otro detalle a resaltar es el acceso a los datos a través del VAP (ver cilindro magenta detrás del VAP), esto se decidió así dadas las capacidades que brinda el VAP de acceso y decodificación de video.
La arquitectura es sencilla pero al mismo tiempo capaz de cumplir con los requerimientos exigidos de una manera flexible y sin restringir mucho al usuario final o al desarrollador de plugins.
En las próximas secciones se describirá con más detalles a los componentes de esta arquitectura, haciendo hincapié en los componentes internos del proyecto.

5.2 Componentes del diseño

5.2.1 iguPlayer

El iguPlayer es el módulo principal o programa anfitrión. Contiene todos los módulos que interactúan con los agentes externos, con los plugins y con el modulo VAP.

Este cuenta con elementos que solucionan y cumplen con los problemas y con las exigencias planteadas. Posee una estructura que soporta plugins, una interfaz gráfica, comunicación con el VAP y permite que tanto los plugins como los usuarios usen sus funcionalidades con facilidad.

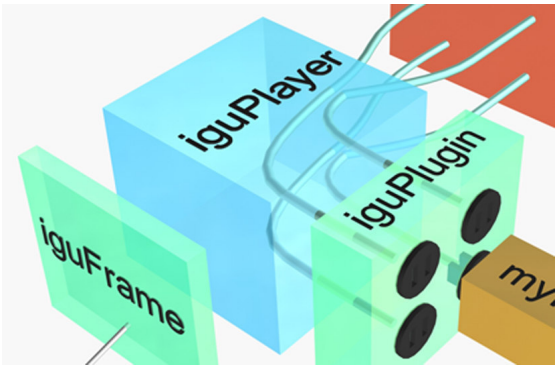


Figura 5-5.- Iguplayer

Estas funciones fueron separadas y asignadas a módulos encargados de realizarlas. Entre estos módulos internos los más importantes son:

- IGU (Sección 5.2.2)
- Manipulador de plugins (Sección 5.2.3.2)
- Interfaz del Programa-Aplicación (Sección 5.2.3.1)

Además de agrupar estos componentes, tiene como función inicializarlos y ponerlos a disposición de los agentes externos. Es por eso que la mejor manera de describirlo es describiendo a esos módulos internos, lo cual se hace a continuación.

5.2.2 IGU (Interfaz Grafica de Usuario)

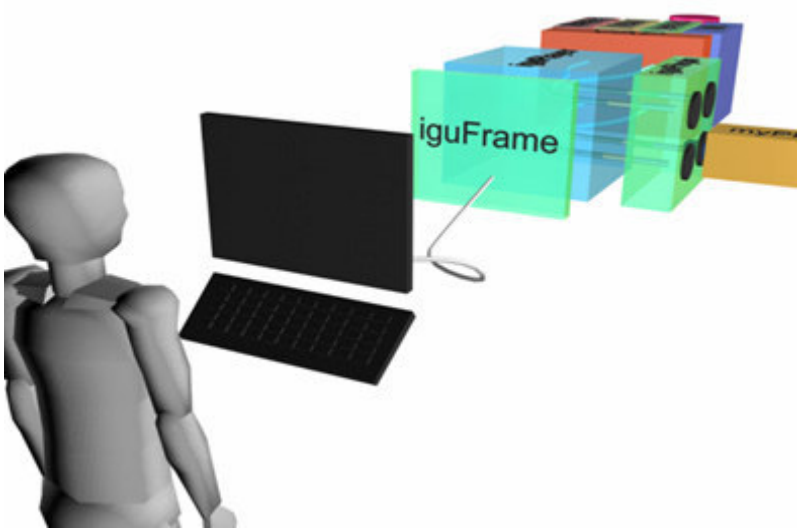


Figura 5-6.- Interfaz gráfica, mostrando

La interfaz gráfica tiene el objetivo, como su nombre lo indica, ser la interfaz entre el programa y el usuario (Figura 5-6). A través de ella es que accederemos a visualizar el procesamiento que realicemos a los archivos de video. La componente fundamental de la misma es la ventana principal, a esta es que se agregan los menús, barras de herramientas, listas, etc. . Tiene como atributos un icono y una barra de estado (Status Bar) a la que agregamos un texto con la versión del programa (Figura 5-7).

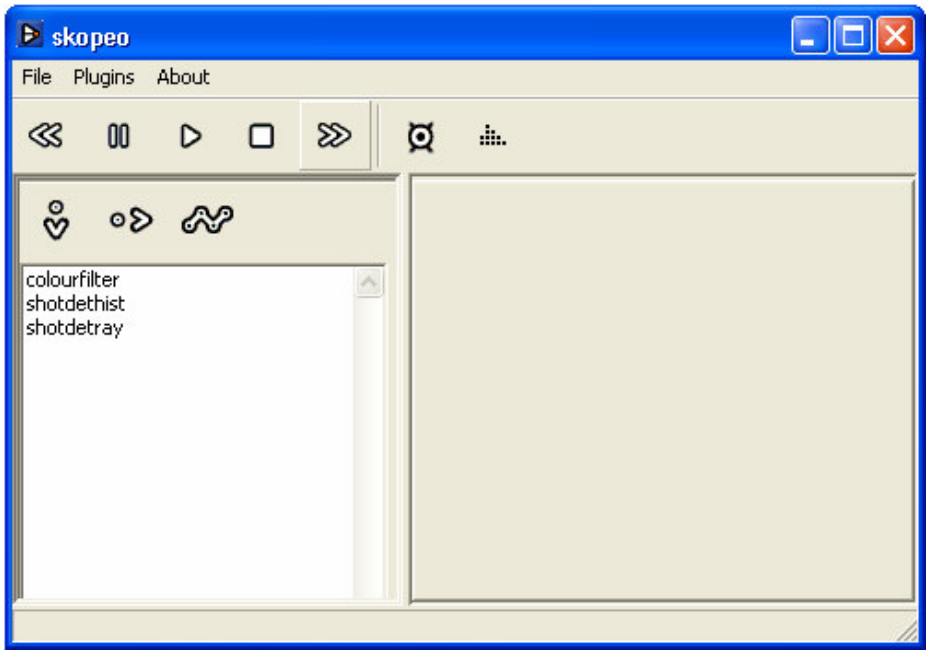


Figura 5-7.- Pantalla principal de la interfaz gráfica de usuario.

Los menús disponibles en la interfaz son los siguientes: Open, Plugin y About (Figura 5.8).

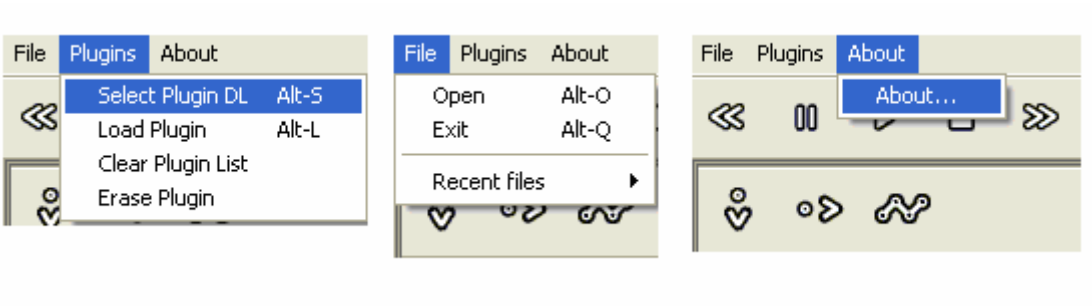


Figura 5-8.- Menús principales.

En el menú Open podemos encontrar las opciones “Open” (abrir archivos), “Exit” (para salir de la aplicación que al seleccionarlo despliega un mensaje pidiendo que confirmemos si queremos salir de la aplicación) y “Recent files” (el cual almacena en submenús los últimos 3 archivos abiertos).

En el menú Plugins podemos encontrar las opciones “Select Plugin DL” (seleccionar plugins), “Load Plugin” (cargar plugins a la lista), “Clear Plugin List” (borrar la lista de plugins) y “Erase Plugin” (borrar la lista de plugins).

En el menú “About” podemos encontrar una breve descripción del programa conteniendo fecha, instituto, nombre de los desarrolladores y versión de la librería.
Cabe señalar que existe la posibilidad de seleccionar algunos submenús desde el teclado simplemente digitando Alt y la letra que corresponda (Ej.: Alt-O para Open).

Contamos con 2 paneles de botones mediante los cuales podemos ejecutar las distintas funciones del programa.

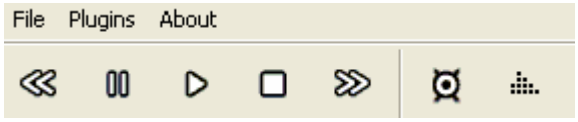


Figura 5-9

El primero contiene todas las herramientas de reproducción de video y además (a la derecha de un separador) cuenta con funciones especiales del VAP (Figura 5-9).

El segundo panel contiene las herramientas de carga, ejecución y concatenación de los plugins (Figura 5-10).

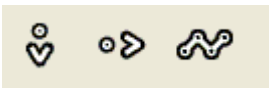


Figura 5-10

En cada caso, la información del botón se exhibe al situar el mouse encima (Figura 5-11).

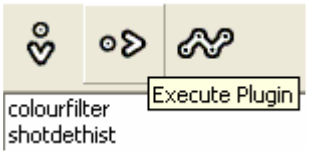


Figura 5-11

Todos los plugins disponibles (presentes en la carpeta “plugins”) se despliegan al iniciar el programa en la lista que vemos en la Figura 5-12. Es posible además extender esta lista para ello debemos ir al menú Plugin como vimos anteriormente.

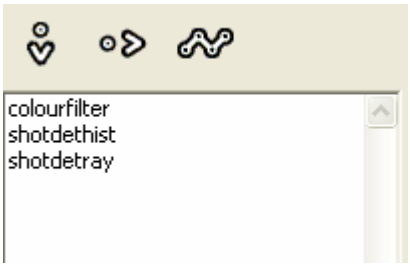


Figura 5-12

Al seleccionar un plugin y ejecutarlo, se despliega el siguiente cuadro:

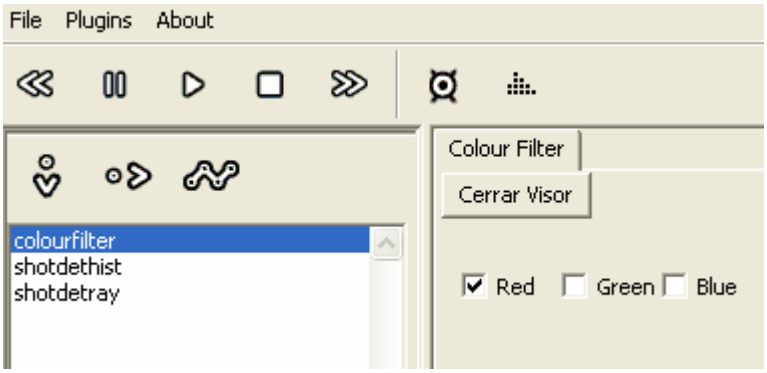


Figura 5-13

En caso de ejecutar otro plugin, se abre una nueva “pestaña” en donde se encuentran datos de entrada o resultados del nuevo plugin ejecutado.

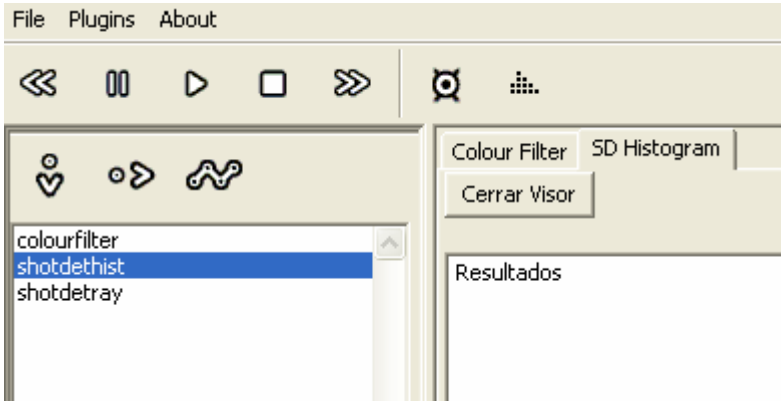


Figura 5-14

La visualización de los videos se realiza a través de una ventana separada de la principal.

Figura 5-15

Lo mismo sucede al ejecutar un plugin, se abre otra ventana (aparte de la que muestra el video) que muestra el resultado de la ejecución del plugin.

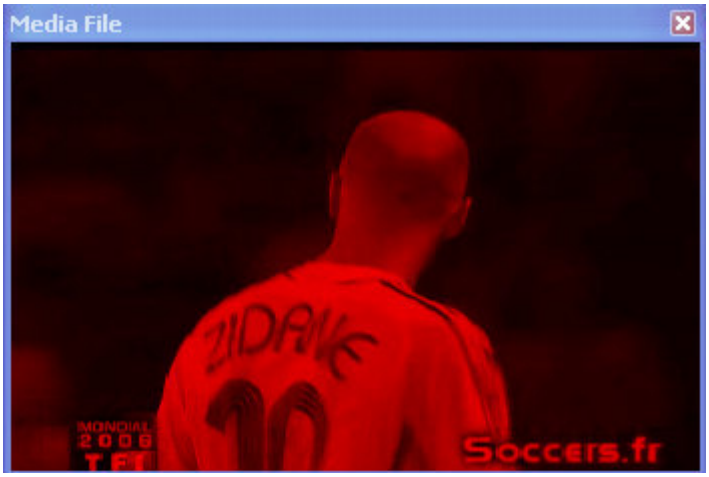


Figura 5-16

5.2.3 Plugins

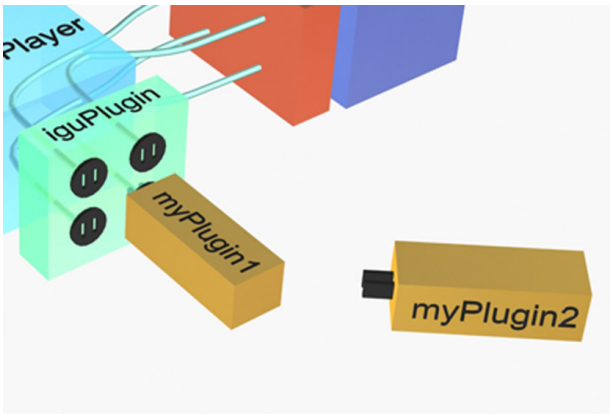


Figura 5-17

Los plugins de un sistema en particular, son extensiones de ese programa que son acoplados al mismo durante el tiempo de ejecución. Con frecuencia se le llama a este programa "anfitrión" porque le brinda al plugin controles, espacio en memoria y acceso a sus funcionalidades.

El propósito y ventaja principal de estos módulos es que son escritos y compilados separadamente del programa anfitrión, y se cargan dinámicamente durante la utilización del mismo.

La arquitectura de plugins fue definida con tres cosas en mente, primero, la necesidad de que se pudiera contar con todas las funcionalidades ofrecidas por el programa padre (iguPlayer, de aquí en más); en segundo lugar, que fuera lo suficientemente simple y completa para permitirle a un desarrollador crear un plugin que logre cumplir con sus necesidades para lograr un testeo de su algoritmo; y en tercer lugar, que se puedan concatenar dos plugins.
Los principales componentes de la arquitectura de plugins son:

- La interfaz
- Manipulador de plugins

Estos dos componentes tienen muy diferentes propósitos, el primero es una definición de todas las cosas que deben implementar todos los plugins para poder comunicarse de manera correcta con el "anfitrión"; mientras que el segundo es el que maneja la creación, el funcionamiento y la destrucción de los plugins.

La arquitectura utilizada está muy extendida en los programas que se encuentran en el ámbito comercial y también en el no comercial, por lo que es importante resaltar la confiabilidad a priori que brinda la misma.

5.2.3.1 Interfaz del Programa-Aplicación (API)

Es el elemento clave en la arquitectura de los plugins. Contiene los métodos que deben implementar todos los plugins para poder comunicarse con el programa anfitrión. El diseño de este elemento se efectuó teniendo en cuenta dos cosas, primero que nada que tuviera todas las cosas necesarias para que se puedan comunicar la mayor cantidad de tipos posibles de plugins, o sea, hacerlo lo más genérico posible dentro de las posibilidades. En segundo lugar se pensó en la comunicación tanto con el VAP, como con otros plugins. La comunicación con otros plugins debió ser tomada en cuenta a la hora de pensar en los componentes necesarios de la interfaz dado que esta sería la misma que se tendría para la comunicación con el VAP, esto nos condicionaba a que la comunicación debía tener muchos puntos en común.

Los elementos más importantes de la interfaz de los plugins desde el punto de vista del diseño de la misma son los siguientes:

- Inicialización de componentes
- Ejecución del plugin
- Destrucción de componentes del plugin
- Seteo de elementos de comunicación
- Identificación del plugin

La idea de la inicialización de componentes es que el mismo plugin inicialice los elementos que utilizará en su ejecución. Se pensó en la inicialización de los diferentes elementos de visualización que necesite el plugin como también por otro lado de los controles del plugin.
La sección de ejecución de plugin es donde efectivamente se efectúa el objetivo principal del proyecto.
Es muy importante que el plugin borre todos los elementos que creó para poder ejecutarse de manera correcta, esto se lleva a cabo en la sección de destrucción de componentes.
Pareció de vital importancia el seteo de diferentes elementos a usar en la comunicación, como los buffers de entrada y salida de datos de los plugins y desde el VAP. Es importante resaltar que estos son los mismos tanto para cuando se comunican entre plugins como cuando se comunican con el VAP.
La identificación de los plugins es un lugar donde se les permite a los desarrolladores dejar diferentes datos sobre el plugin que han creado. Esto está orientado a que exista una breve cantidad de información útil para los usuarios.

5.2.3.2 Manipulador de Plugins

El manipulador de plugins es un módulo que existe dentro del programa anfitrión que regula y maneja, valga la redundancia, el uso de los plugins. Administra el acoplamiento, desacoplamiento, ejecución, almacenamiento y orden de los plugins. Inicializa todas las variables que los plugins necesitan para ejecutarse, tales como direcciones de memoria de donde obtener datos y donde colocarlos (buffers de entrada y salida), banderas de escritura y lectura de información, etc.

El manipulador se diseñó de manera que contenga una colección de plugins que posibilite al usuario la sencilla selección del plugin a usar. Además de esto el manipulador está preparado para cargar nuevos plugins, crear e inicializar todos los elementos que necesiten, destruir los elementos creados y descargar los plugins de la lista.

También tiene como responsabilidad la concatenación de plugins. Dados dos plugins cualquiera, el usuario podrá concatenarlos con la ayuda del mismo.

5.3 Herramientas

La herramienta más importante y casi única que se usa en este proyecto además del mismo lenguaje y sus compiladores, es la librería multiplataforma de C++ wxWidgets. Esta librería ha demostrado ser de mucha utilidad a la hora del desarrollo del programa, mostrando ser simple de usar, con muchas prestaciones, buena documentación y buen rendimiento en runtime. Estas características llevaron a confiar en ella en el momento de elegirla para manejar la arquitectura de plugins.
A continuación se efectúa un recorrido por las características generales de la librería y por las características particulares de los elementos más importantes de la implementación del programa.

5.3.1 WxWidgets

WxWidgets es una librería multiplataforma para diseño de GUIs, además de esto tiene otras herramientas como clases para cargar librería dinámicas, clases para reproducción de audio y video, etc.
Las plataformas que soporta wxWidgets son, MS Windows, MacOS, Unix con GTK+, Unix con Motif, etc. En este caso lo que interesa es que soporte sistemas Unix y Windows.
Las herramientas de wxWidgets más usadas y más importantes para el programa son las clases wxFrame, wxBitmap, wxClientDC, wxImage, wxDynamicLibrary y wxArray.
Estas clases son las que resaltan por la importancia que tienen en la implementación del programa, se pueden separar en tres usos, wxFrame es la clase madre de todo lo que es interfaz gráfica, wxBitmap, wxClientDC y wxImage son las clases que se utilizan para poder mostrar en pantalla tanto los frames de un video como un histograma; y por último wxDynamicLibrary y wxArray son parte importante de la implementación de la arquitectura de plugins.

5.3.1.1 Interfaz gráfica

WxWidgets es una librería pensada para la implementación de interfaces gráficas, por lo que contiene sencillamente todas las herramientas básicas que puede precisar una interfaz, léase menús, sliders, barras de herramientas, cuadros de diálogo, etc.
Existen una cantidad de clases en wxWidgets que sirven para la implementación de estas herramientas básicas, pero hay una clase que sobresale de las demás por ser generalmente la que usan los demás elementos de una interfaz gráfica, esta clase es wxFrame.
WxFrame es una clase de ventanas que deriva de otras clases tales como wxObject y wxWindow, en este proyecto la ventana principal del programa es de la clase wxFrame, esto le permite poder contener casi cualquier elemento de los que ofrece wxWidgets exceptuando a otro wxFrame o un cuadro de diálogo.
El constructor de la clase permite seleccionar diferentes opciones acerca del tipo de frame, su tamaño, forma, etc.

```
wxFrame (wxWindow* parent, wxWindowID id, const wxString& title, const wxPoint&
pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
 long style = wxDEFAULT_FRAME_STYLE, const wxString& name = "frame")
```

5.3.1.2 Herramientas usadas en la arquitectura de plugins

Los elementos más importantes de wxWidgets que se usan para la implementación de la arquitectura de plugins son las clases wxDynamicLibrary y wxArray, la primera para cargar y manejar las librerías dinámicas, las segunda para manejar una lista de librerías que están cargadas en el programa y pueden ser usadas al ser seleccionadas.

5.3.1.2.1 WxDynamicLibrary

Esta es una clase que representa una librería dinámica (dll en Windows, shared library en Unix, etc.). Para cargar una librería solamente se tiene que crear un objeto de esta clase, luego esta clase ofrece métodos como Load (carga la librería dinámica), Unload (descarga la librería dinámica), GetSymbol (devuelve un puntero al símbolo que se le pasa como variable o NULL si no encuentra el símbolo en la librería), etc.; estos métodos permiten manejar la librería creada.

5.3.1.2.2 WxArray

WxArray ofrece la posibilidad de generar un array dinámico de cualquier tipo de objeto o estructura siempre y cuando se use el wxObjArray (es el que usamos en el programa), dado que este tipo de arrays dinámicos tratan a sus elementos como objetos.
WxArray ofrece un diseño relativamente eficiente [2], tanto en tiempo de ejecución como en uso de memoria y tamaño del ejecutable. El tiempo de acceso a un miembro es constante, manejan automáticamente la utilización de memoria y hasta se puede chequear si los índices están correctos, o sea, no se podrá acceder a un elemento con un índice incorrecto.

5.4 Implementación

A continuación se describirá la implementación de los principales módulos que componen el sistema diseñado y la comunicación entre ellos.

5.4.1 Aplicación Principal

Como mencionamos anteriormente, el componente central de nuestro proyecto es el iguPlayer. Esta conformada por una clase del mismo nombre que además deriva de la clase de wxWidgets wxApp. Representa a la aplicación misma y cualquier interfaz grafica basada en wxWidgets debe poseer una. Con ella se establecen y obtienen las propiedades de ancho y alto, el sistema de mensajes de ventanas, iniciación y finalización de la aplicación y procesamiento de eventos [7].

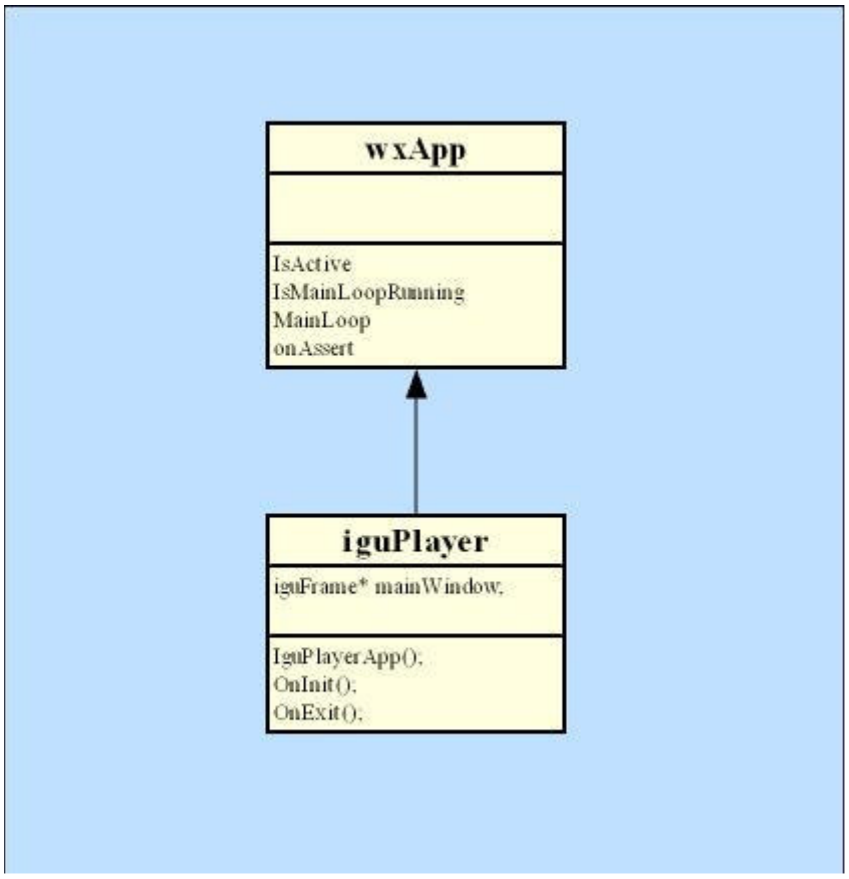


Figura 5-18.

Los métodos principales que iguPlayer implementa de wxApp son

- OnInit(): llamado en la iniciación del programa. Crea la ventana principal, inicia los manipuladores de bitmaps y de eventos, carga las imágenes para los iconos, etc.

- OnExit(); llamado en la finalización. Destruye las ventanas y los controles cuando la finalización del programa es invocada.

También contiene la ventana principal (mainWindow), en donde se encuentran todas las otras ventanas y controles gráficos.

Si bien iguPlayer interactúa con otras clases del sistema, las mismas también derivan de otras clases madres de wxWidgets por lo tanto, incluiremos mas diagramas por cada clase restante. El primero ilustrara de que clase de wxWidgets deriva, y otro(s) para describir la interacción entre las demás clases del sistema. A medida que se vayan describiendo las clases restantes, se agregaran los diagramas que ilustran la comunicación.

5.4.2 Ventana Principal

La ventana principal esta implementada por la clase iguFrame. La misma deriva de la clase wxFrame de wxWidgets, por lo tanto sobrescribe algunos de sus métodos y contiene otros exclusivos, así como atributos exclusivos. Estos atributos son los controles, ventanas y objetos que componen el resto del sistema.

Cualquier acción que el usuario de Skopeo quiera tomar, lo hará a través de un control de la clase iguFrame. Desde seleccionar un video o un plugin, hasta finalizar la aplicación. Todas las ventanas extras en las cuales se despliegan los cuadros, pertenecen también a esta clase.

- OnOpenFile: Seleccionar un archivo de video
- OnExit: Finalizar la aplicación
- OnLoadDL: Cargar una librería dinámicamente
- OnPause: Detener la reproducción del video
- OnLoadPI: Cargar un plugin
- OnPluginExec: Ejecutar un plugin
- OnConcPI: Concatenar plugins

Son algunos de los métodos que se utilizan para estas acciones.

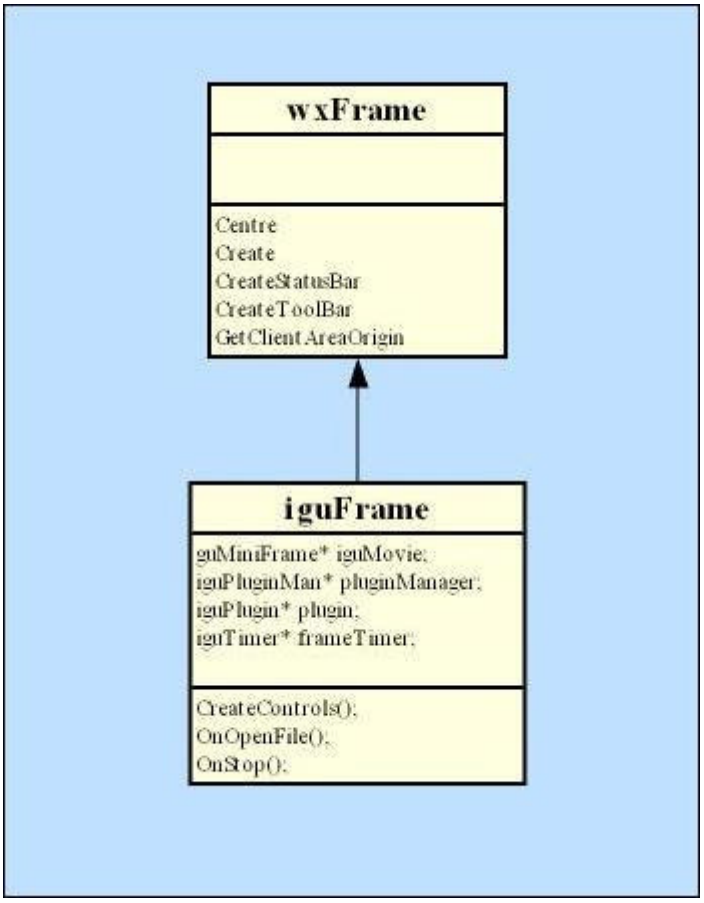


Figura 5-19

Sin embargo, el método fundamental de esta clase es CreateControls. Invocándolo se crean los demás controles del sistema, barras de herramientas, menús, botones, cuadros de dialogo.
También se inicializan los objetos de las restantes clases que componen el sistema, como por ejemplo el manipulador de plugins, la colección de plugins, el visor de video etc. Los mismos son declarados en el archivo de encabezado e inicializados en este método.

Otros métodos que esta clase posee (mencionados antes del diagrama) son los que se invocan al ejecutar los botones u opciones de menú que se despliegan en la ventana. La reproducción de un video, la selección de los archivos de plugins o el video mismo, el avance cuadro a cuadro son algunos ejemplos de los métodos que se implementan en esta clase.

En lo que respecta a la ejecución de los plugins, los métodos de esta clase no son los que efectivamente los manipulan, si no los que llaman a los métodos del manipulador de plugins, descrito mas adelante.

iguFrame posee dos atributos que juegan un papel importante en la comunicación entre los plugins y el anfitrión. Son dos bufferes de entrada y salida de datos (mainBufferIn y mainBufferOut) y con ellos se conectan los plugins (a nivel de transmisión de información) con el sistema principal. A su vez, los plugins también poseen bufferes, pero esto se explicara mas adelante.

iguFrame también cumple un rol clave en la comunicación del VAP con el resto de la aplicación. A través de esta clase es que se accede directamente a sus funcionalidades para elegir el archivo a procesar, y seleccionar las herramientas de manipulación de datos que sean de interés. Mas adelante se explicara con detalle como se logro esta comunicación.

5.4.3 Manipulador de plugins

Este modulo se implementa mediante la clase iguPluginMan. Esta clase deriva de la clase exEvtHandler de wxWidgets. Si bien todos los métodos métodos y atributos son exclusivos de ella, es decir, no son sobrescritos de wxEvtHandler, se necesito que derivara de esta para poder administrar los cronómetros que regulan el despliegue de los cuadros en el visor de video.

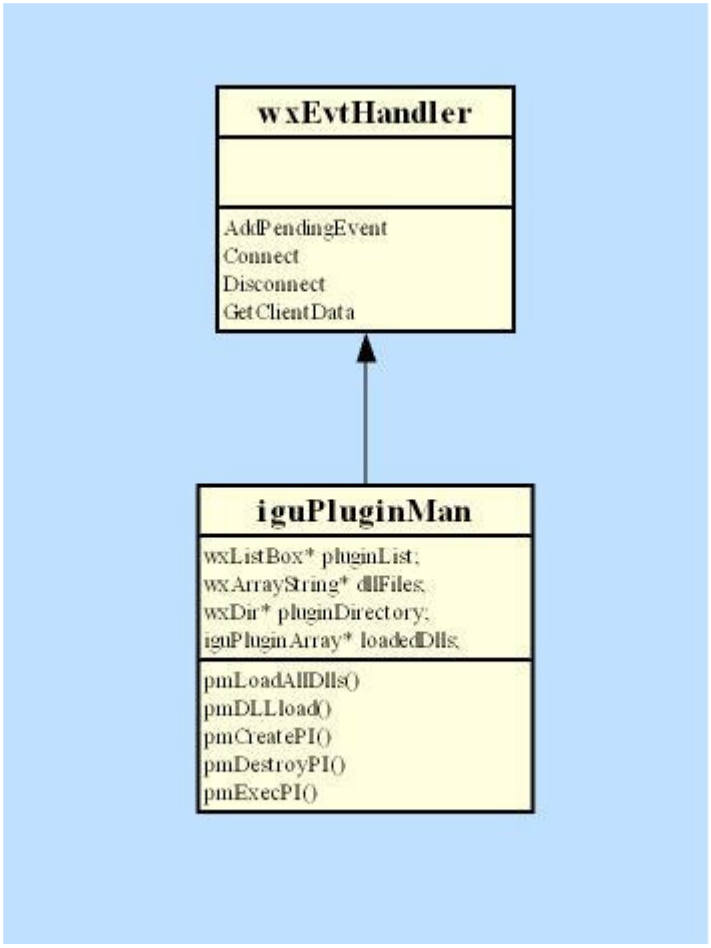


Figura 5-20

La clase iguPluginMan contiene los métodos necesarios para administrar y manipular los plugins. Hay métodos para cargar las librerías y almacenarlas en una colección, métodos para cargar el plugin, ejecutarlo, destruirlo etc. Entre ellos los más importantes son:

- pmLoadAllDlls: método que cargar todas las dll's del directorio de plugins
- pmDLLload: método de carga de .dll
- pmCreatePI: método que crea el objeto del plugin
- pmDestroyPI: método que borra el objeto creado, destructor
- pmExecPI: método de ejecución de plugin seleccionado

El primero se invoca al inicializarse la aplicación, y carga todas las librerías dinámicas que se encuentran en la carpeta plugins. Dichas librerías se cargan una a una mediante el método pmDLLload y se van almacenando en la colección de librerías (iguPluginArray, explicado mas adelante).

Cuando el usuario quiere cargar un plugin de alguna de las librerías de la colección, la selecciona de la lista desplegada en iguFrame y se llama al método pmCreatePI. Este crea el plugin y lo almacena en el iguPluginArray, en el lugar para plugins correspondiente a la librería elegida.

5.4.4 Colección de plugins

La colección de plugins se implementa con dos clases: iguPluginArray e iguPluginContainer.

iguPluginContainer es una clase que posee dos atributos: un objeto de la clase wxDynamicLibrary el cual se crea para cargar una librería dinámica, y un objeto de la clase iguPlugin, el cual es la clase madre de todos los plugins para este sistema.

iguPluginArray deriva de la clase wxArray de wxWidgets, el cual implementa un arreglo de objetos del tipo que se quiera. Se construye mediante macros y tiene métodos básicos de manipulación de arrays, los cuales son utilizados por el iguPluginMan para administrar la colección de plugins. Entre ellos se encuentran [7]:

- Add: Agrega un elemento al array
- Clear: Vacía el array de elementos y libera memoria
- Item: Devuelve el objeto del índice deseado
- RemoveAt: Elimina elementos del array.

Almacenar los plugins en este arreglo permite manipularlos por separado y acceder a sus respectivos métodos independientemente de la cantidad que se hayan creado.

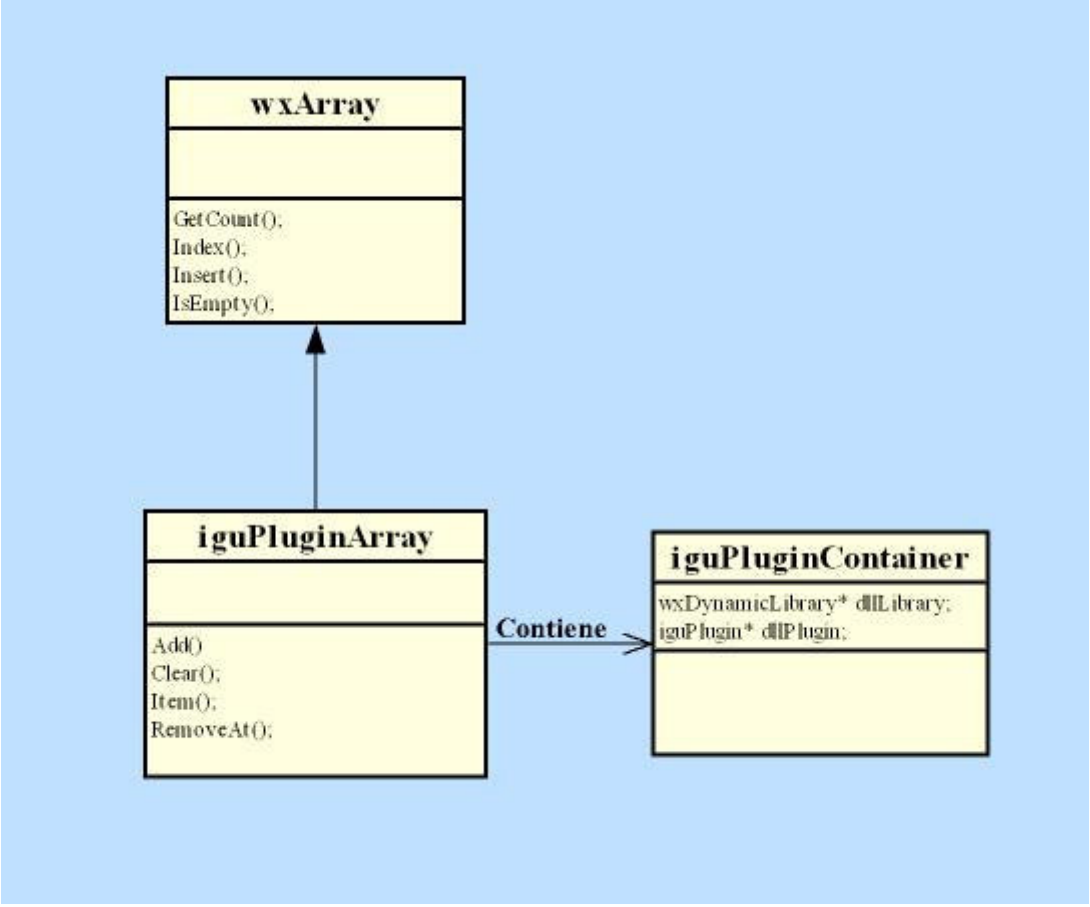


Figura 5-21

5.4.5 Interfaz del Programa-Aplicación (API)

Es el componente central en la arquitectura de plugins. Esta implementado por la clase abstracta iguPlugin y de esta interfaz heredan todos los plugins de este sistema. En esta clase se declaran todo los métodos que implementaran luego los plugins, y es a través de ella que el programa anfitrión puede acoplarlos, almacenarlos, ejecutarlos y destruirlos.

Los principales son:

- initPlugin: Método que inicia todas las variables del plugin
- loadControls: Carga los controles
- execPlugin: Ejecución del plugin
- destroyPlugin : Destrucción del plugin
- setBufferIn/Out: Seteo de los buffers de entrada/salida
- getBufferIn/Out: Obtención del buffer de entrada /salida

Ejemplos de algunos de estos metodos se presentan mas adelante en el capitulo 6 Plugins.

También posee los atributos principales para la transmisión de datos desde el al anfitrion a los plugins, y entre los plugins mismos. Cuatro elementos claves en la transmisión y recepción de datos son:

- bufferIn: Bufferes de entrada y de salida
- bufferOut:
- bool dataInReady: Banderas de entrada y salida de datos
- bool dataOutReady:

Sin embargo, el elemento esencial de la arquitectura, es el punto de entrada (entry point) que aporta esta clase a los plugins, para que puedan “soldarse” al sistema anfitrion. El mismo esta compuesto por dos funciones con propositos diferentes.

La primera es CreateIguPlugin(), la cual se exporta desde la librería dinamica hacia el anfitrion de la siguiente manera:

```
extern "C" __declspec(dllexport) iguPlugin* CreateIguPlugin(){return new name; }
```

El anfitrion carga el plugin al crear una instancia de una clase que deriva de la interfaz invocando a esta funcion.

La segunda es una funcion estandar de inicializacion de la librerías dinamicas:

```
BOOL APIENTRY DllMain( HANDLE hModule, DWORD ul_reason_for_call,
                                       LPVOID lpReserved)
```

Habiendo descrito las clases iguFrame, iguPluginMan, iguPluginArray e iguPlugin podemos ilustrar mediante un diagrama de clases que los incluya a todos, como se comunican unos con otros.

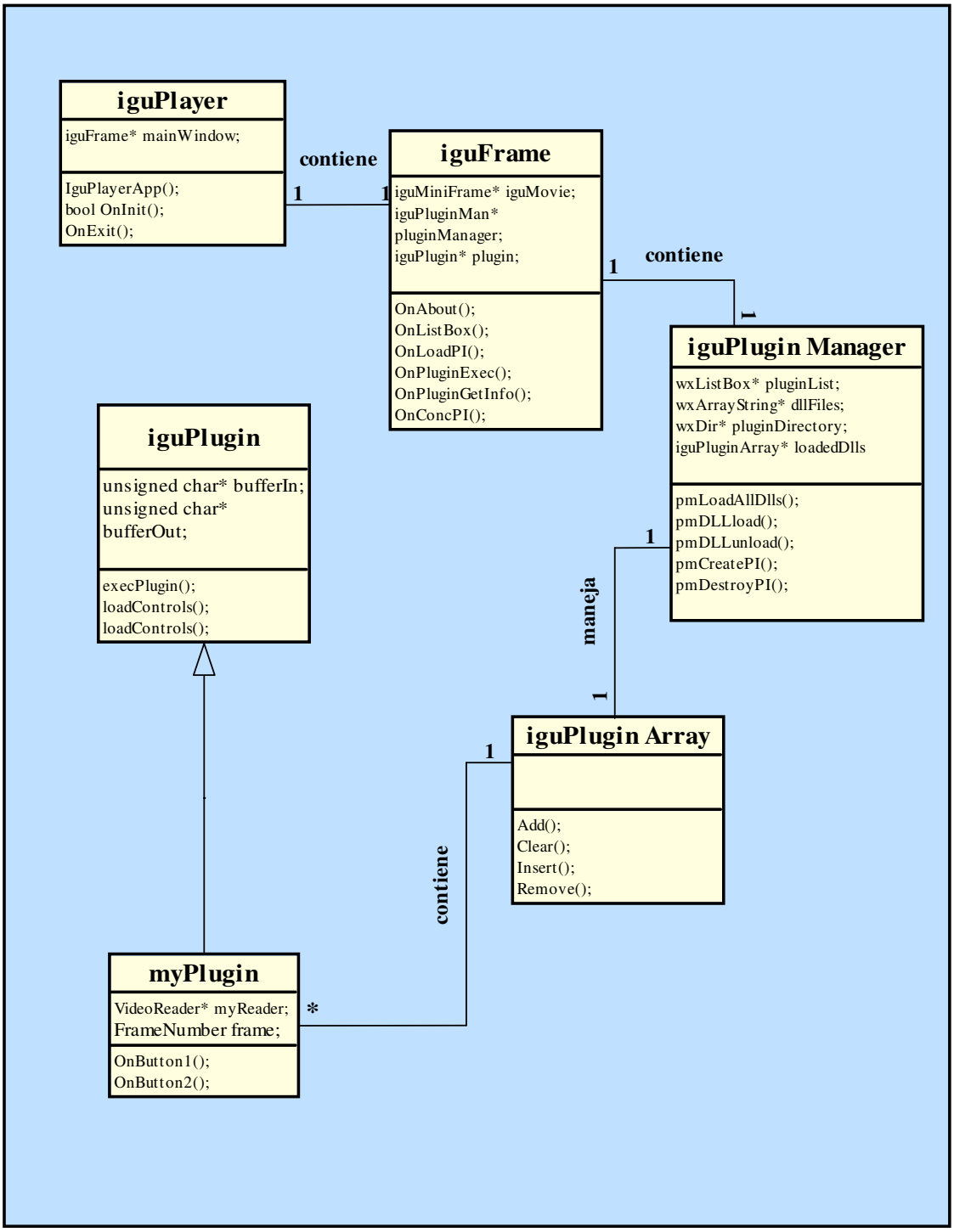

iguPlayer
iguFrame* mainWindow;
IguPlayerApp();
bool OnInit();
OnExit();
contiene
1
1
iguFrame
iguMiniFrame* iguMovie;
iguPluginMan* pluginManager;
iguPlugin* plugin;
OnAbout();
OnListBox();
OnLoadPI();
OnPluginExec();
OnPluginGetInfo();
OnConcPI();
1
contiene
1
iguPlugin Manager
wxListBox* pluginList;
wxArrayString* dllFiles;
wxDir* pluginDirectory;
iguPluginArray* loadedDlls
pmLoadAllDlls();
pmDLLload();
pmDLLunload();
pmCreatePI();
pmDestroyPI();
iguPlugin
unsigned char* bufferIn;
unsigned char* bufferOut;
execPlugin();
loadControls();
loadControls();
1
maneja
1
iguPlugin Array
Add();
Clear();
Insert();
Remove();
1
contiene
myPlugin
VideoReader* myReader;
FrameNumber frame;
OnButton1();
OnButton2();
*

Figura 5-22

5.4.6 Visor de Video

El usuario puede visualizar el resultado de su trabajo mediante ventanas que despliegan la información de video. Esas ventanas son implementadas por la clase iguMiniFrame y derivan de la clase wxMiniFrame de wxWidgets. En ella se "pinta" el cuadro con los datos entregados por el sistema principal. Los métodos principales de esta clase son los siguientes:

- displayData: Despliega los datos en la ventana
- Play: Prolonga el despliegue de datos en el tiempo
- getDataFromHost: Lee los datos que el anfitrión coloco en su buffer de salida.

La prolongación del despliegue de los cuadros en el tiempo es lo que da la sensación de continuidad, y es lo que constituye la reproducción de un archivo de video.

El control del tiempo se implementa mediante la clase iguTimer, la cual hereda de la clase wxTimer de wxWidgets. Sus métodos principales son

- SetOwner: Se establece el "dueño" del cronometro. Explicado más adelante
- Start: Inicialización del cronometro
- Stop: Detenimiento.

Al invocar SetOwner (invocado por iguFrame), se le asigna al cronometro un objeto de la clase wxEvtHandler (o cualquiera que herede de ella) el cual deberá ser su dueño. Este dueño recibirá eventos periódicamente con números de identificación de parte de el cronometro, los cuales accionaran sus métodos. En el caso del iguMiniFrame, el método periódicamente accionado es Play. Play llamara a DisplayData y así la película se reproducirá.

Habiendo descrito las clases iguFrame, iguMiniFrame, e iguTimer podemos ilustrar mediante un diagrama de clases que los incluya a todos, como se comunican unos con otros.

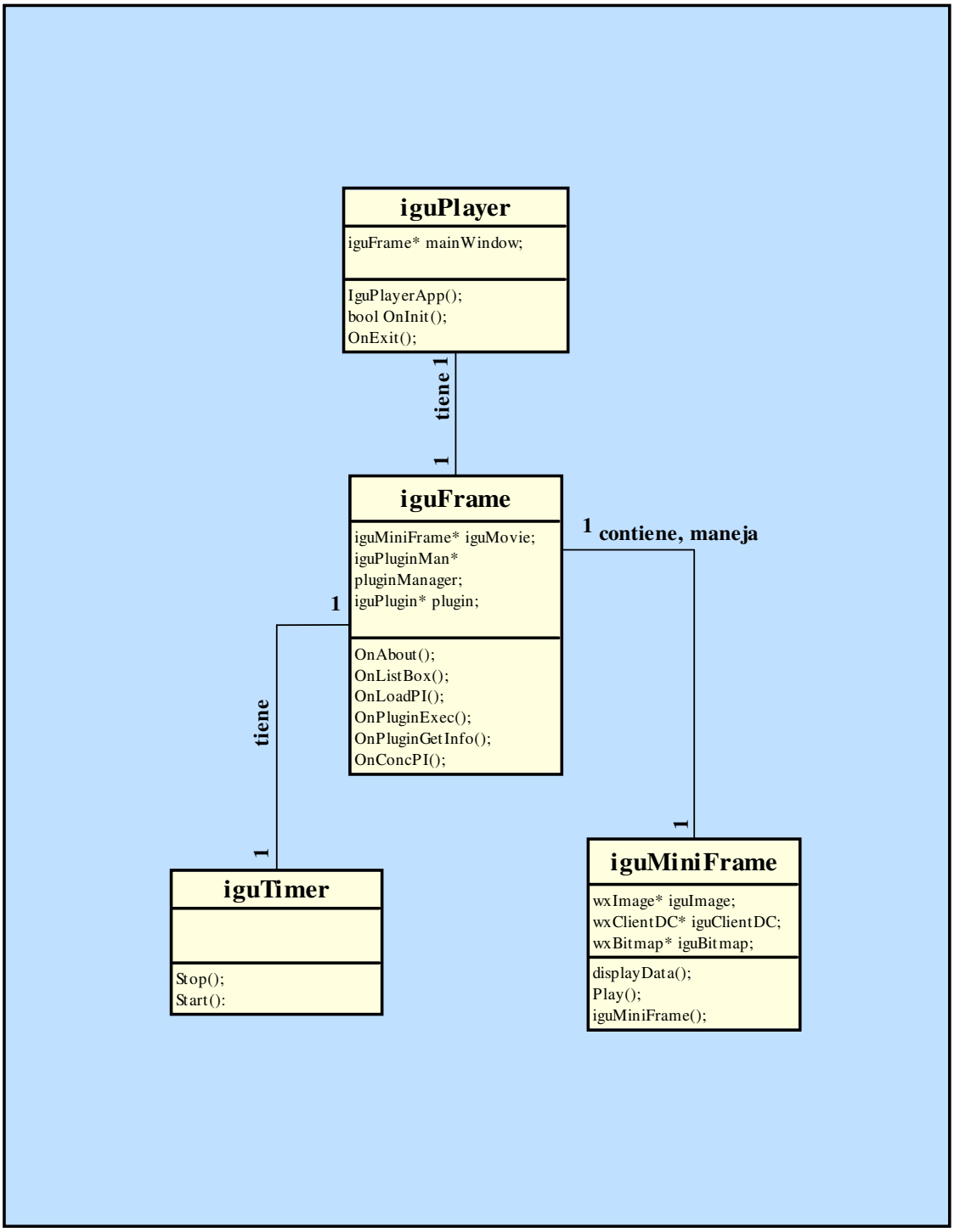

iguPlayer
iguFrame* mainWindow;
IguPlayerApp();
bool OnInit();
OnExit();
tiene 1
1
iguFrame
iguMiniFrame* iguMovie;
iguPluginMan*
pluginManager;
iguPlugin* plugin;
OnAbout();
OnListBox();
OnLoadPI();
OnPluginExec();
OnPluginGetInfo();
OnConcPI();
1 contiene, maneja
1
tiene
1
iguTimer
Stop();
Start():
1
iguMiniFrame
wxImage* iguImage;
wxClientDC* iguClientDC;
wxBitmap* iguBitmap;
displayData();
Play();
iguMiniFrame();

Figura 5-23

5.4.7 Comunicación plugins-anfitrión

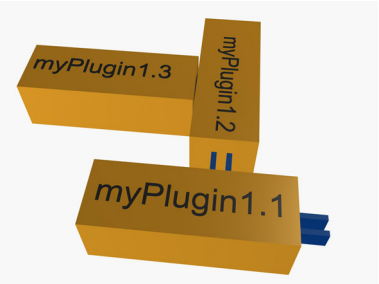


Figura 5-24

La comunicación entre los plugins y el programa anfitrión a nivel de transmisión de datos se realiza mediante los buffers de entrada/salida y banderas de indicación de datos listos para ser tomados o entregados (ver Figura 1.4.7).

Del lado del anfitrión, los métodos principales involucrados en esta comunicación son los del iguPluginManager, sin embargo el usuario accede a algunos de ellos mediante el iguFrame.

Como especificamos en los subcapitulos de esta sección, tanto el iguFrame como los plugins poseen bufferes de entrada o salida. Esto hace posible que el plugin tome datos del anfitrión, los manipule y devuelva al anfitrión o, a otro plugin, para que este a su vez los manipule y entrega a quienquiera se le haya asignado como destinatario.

El "destinatario" no es mas ni menos que otro plugin o el host mismo, solo que su buffer de entrada (BI) será el buffer de salida (BO) del remitente, por así decirlo. De esta forma se implementa la concatenación de los plugins, y los resultados pueden ser desplegados en el anfitrión (a través de las ventanas iguMiniFrame) o guardarse en memoria etc. .

Este proceso se realiza al llamar al método getBufferOut de un plugin, este retorna el puntero al espacio de memoria de su buffer de salida, y al llamar al método setBufferIn del plugin remitente pasándole el buffer de salida del puntero destinatario, se establece la concatenación.

En la figura 5-25 se ve un diagrama que describe cómo los plugins y el anfitrión quedan concatenados:

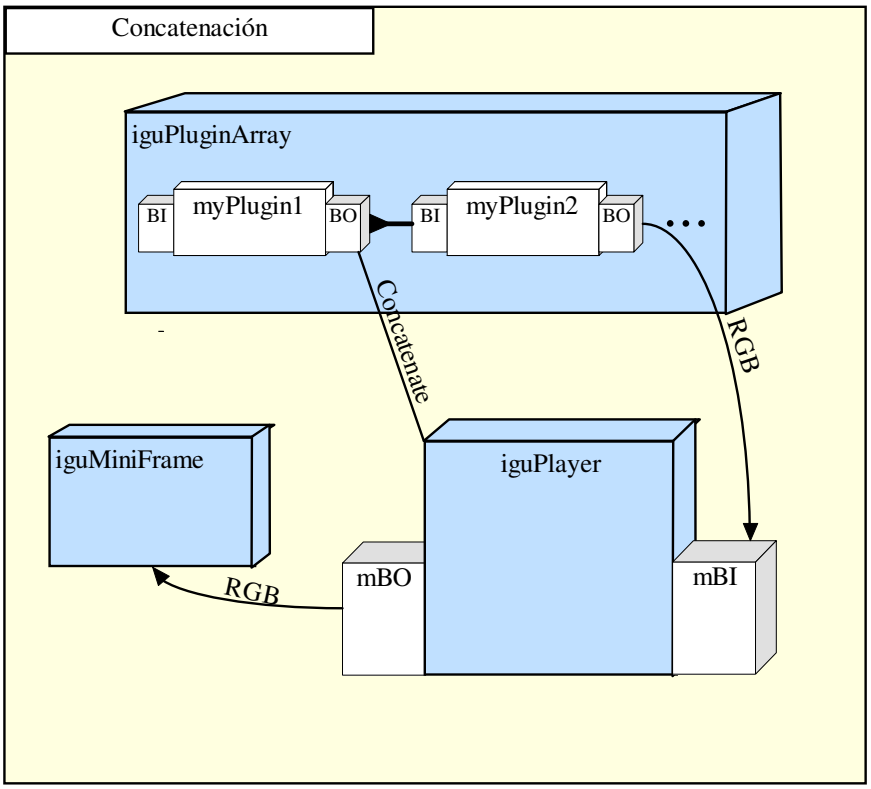


Figura 5-25.- Diagrama de concatenación. (BI/O: Buffer de entrada/salida; mBI/O: buffer de entrada/salida del anfitrión).

6 PLUGINS

En esta sección se mostrarán y comentarán algunos plugins que se implementaron para probar la aplicación. Primeramente se verá un plugin de tipo filtro, que recibe como parámetro un cuadro y una componente (R, G o B) y devuelve el mismo modificado de acuerdo al parámetro que le hayamos pasado. Luego se verá un plugin del tipo detector de eventos, el cual analiza el frame actual con respecto a los anteriores y detecta un cambio de escena.

Estos dos ejemplos representan dos tipos diferentes de plugins dado que uno afecta los datos directos del video y nos devuelve datos procesados (filtro), mientras que el otro nos devuelve como resultado otro tipo de información; el cuadro en el que se da el cambio de escena(detector de eventos).

6.1 ColourFilter

Este plugin es un filtro que permite quitar las componentes fundamentales RGB (Red, Green y Blue; Rojo, Verde y Azul) de un video y visualizar el resultado (Figura 1).

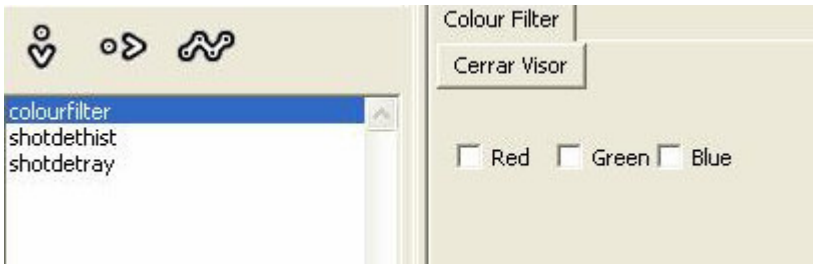


Figura 6-1.- Vista del visor del plugin ColourFilter

Se compone fundamentalmente de los métodos definidos en la interfaz para plugins que son:

- initPlugin
- loadControls
- execPlugin

initPlugin:

Recibe como parámetros de entrada un objeto de la clase wxWindow (una ventana) y otro de la clase wxChar (un videopath). Retorna un panel.

Básicamente se encarga de la inicialización. Esto es inicializar los handlers de imagen, la creación de la instancia de VideoReader y el panel donde se insertarán los controles. En este método podemos apreciar como los plugins acceden a las funcionalidades del vap (Ej., VideoReader)

```
wxPanel* colourFilter::initPlugin(wxWindow* parent, wxChar* videoPath)
{
wxInitAllImageHandlers();       // inicializo los handlers de
                                // imagen
        VideoReader::SetVideoFileName(videoPath);    // obtengo la instancia
                                                     // de Videoreader
        myReader=VideoReader::GetInstance();
        p2 = new wxPanel(parent,-1);              // creo el panel donde se insertaran
                                                  // los controles.
```

```
        dataOutReady=false;
        bufferIn = new unsigned char;
        bufferOut = new unsigned char;
        loadControls();
        return p2;
}
```

loadControls:

No recibe ni retorna ningún parámetro.

Carga todos los controles necesarios para el manejo y el funcionamiento del plugin. Esto es los checkboxs que se usan para seleccionar qué color se quiere quitar del video, el botón para cerrar el visor y el visor de los resultados del plugin.

execPlugin:

No recibe ni retorna ningún parámetro.

Este es el método principal, en él se ejecuta el filtrado efectivo del video. Cada vez que pide un nuevo frame, verifica cuales son los colores que se seleccionaron para filtrar y de acuerdo a esto lleva la componente correspondiente a cero. Luego de efectuar el filtrado, se colocan los datos en el buffer de salida y en el visor del plugin. Cabe destacar que este ejemplo demuestra como los plugins pueden acceder a los datos del video independientemente del anfitrion.

```
void colourFilter::execPlugin()
{
        dataOutReady=false;
        bufferIn = myReader->GetNextRGB();  //Acceso a los datos del
                                                        //video.
        int width = myReader->GetWidth();
        int height = myReader->GetHeight();

        for(int i=0;i<width*height;i++){
        if(red->IsChecked())
                bufferIn[i*3]=0;    // Red
        if(green->IsChecked())
                bufferIn[i*3+1]=0;  // Green
        if(blue->IsChecked())
                bufferIn[i*3+2]=0;  // Blue
        }
        bufferOut=bufferIn;      //Colocación de datos en el buffer de
                                                        // salida
        dataOutReady = true;
        // Aquí se despliegan los datos RGB en el visor:
        /*********************************************/
        MyImage = new wxImage(width,height,bufferIn,false);
        MyBitmap = new wxBitmap(MyImage,-1);
        MyClient = new wxClientDC(viewer);
        MyClient->DrawBitmap(*MyBitmap,0,0,false);
        /*********************************************/
        frame++;
}
```

Resultados de la aplicación del plugin ColourFilter:

Imagen normal

Imagen sin componente R

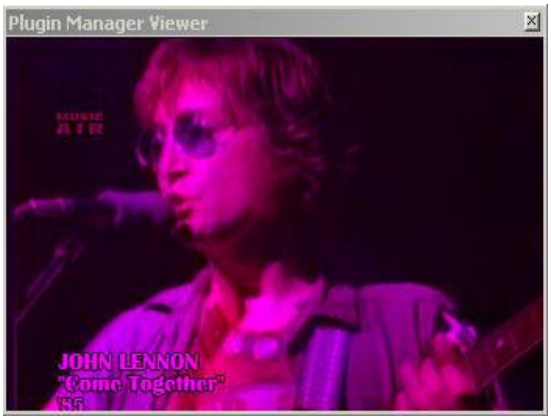


Imagen sin componente G

Imagen sin componente B

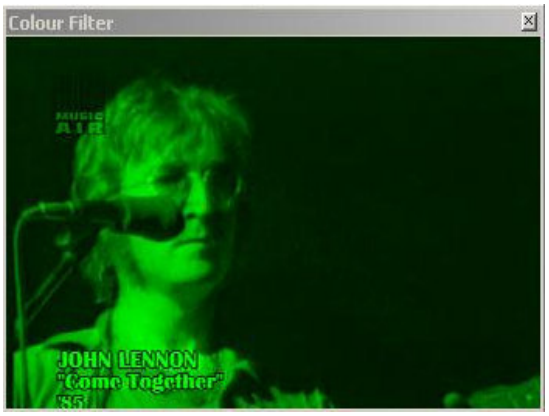


Imagen sin componente R y B

Imagen sin componente G y B

6.2 Detección de cambio de escena (ShotDetector)

Este plugin utiliza las diferentes herramientas que provee el VAP para detectar el cuadro en el cual se efectuó un cambio de escena en el archivo de video cargado.

Se compone fundamentalmente de los métodos definidos en la interfaz para plugins que son:

- initPlugin
- loadControls
- execPlugin

initPlugin:

Recibe como parámetros de entrada un objeto de la clase wxWindow (una ventana) y otro de la clase wxChar (un videopath). Retorna un panel.

Básicamente se encarga de la inicialización. Esto es inicializar los handlers de imagen, la creación de la instancia de VideoReader, ShotDetectoRayleigh y el panel donde se insertarán los controles.

```
wxInitAllImageHandlers();                  //inicializo los handlers de imagen

VideoReader::SetVideoFileName(videoPath);  // obtengo la
                                           //instancia de VideoReader
myReader = VideoReader::GetInstance();
piSDR = new ShotDetectorRayleigh();        // creo el objeto de
                                           // ShotdetectorRayleigh
```

loadControls:

No recibe ni retorna ningún parámetro.

Carga todos los controles necesarios para el manejo y el funcionamiento del plugin. Esto es los checkboxs que se usan para seleccionar qué color se quiere quitar del video, el botón para cerrar el visor y el visor de los resultados del plugin.

execPlugin:

Llama al método **Analyzeframe** de VAP que recibe como parámetro un cuadro, si la variable retornada indica que fue un cambio de escena, agrega al visor la frase “Shot Change”, sino agrega “Same Shot”.

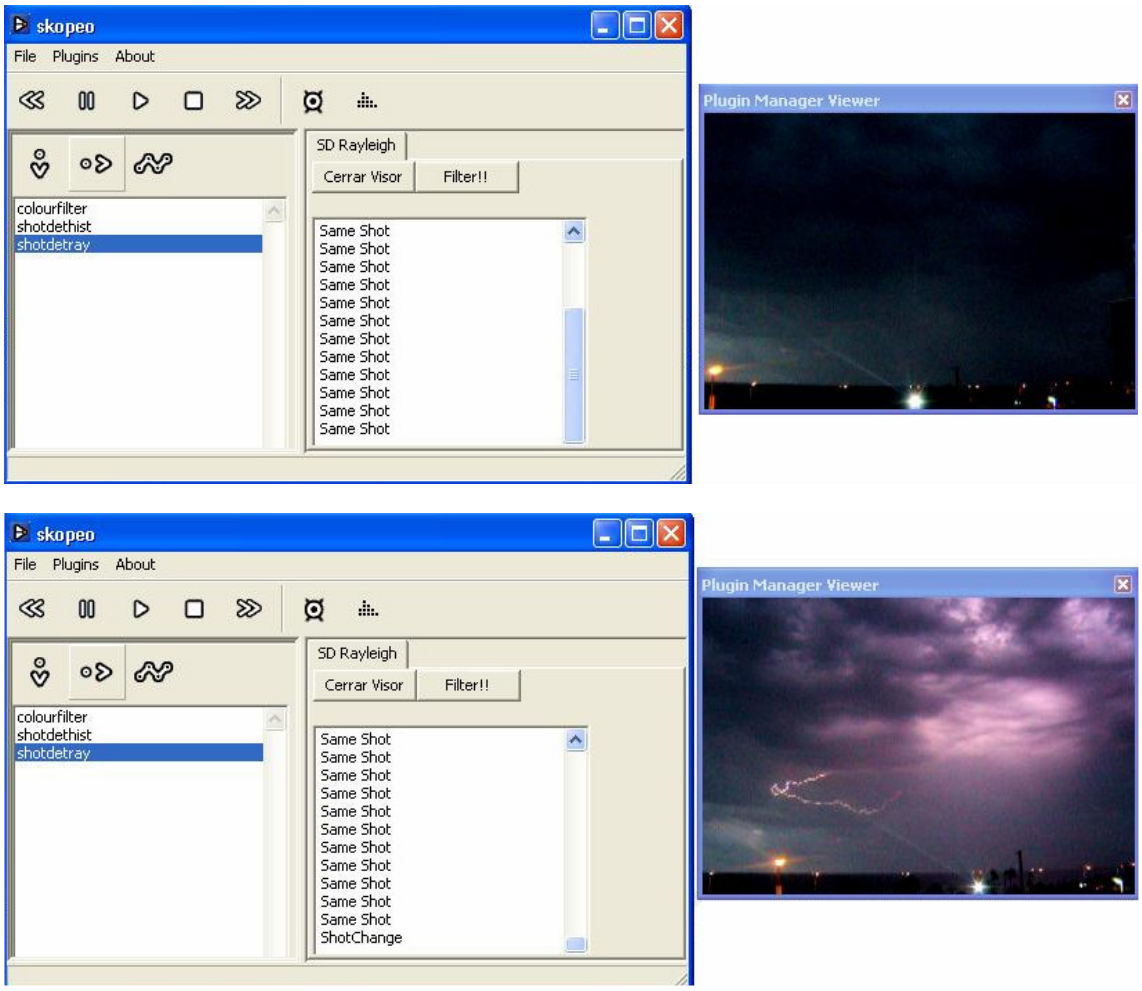
Ese proceso puede verse en el siguiente extracto del método:

```
bufferIn = myReader->GetNextRGB();
result= piSDR->AnalyzeFrame(frame);

if(result->shotChange)
{
        resultView->AppendText("ShotChange\n");
}
else
{
        resultView->AppendText("Same Shot\n");
}
```

No recibe ni retorna ningún parámetro.

Resultados de la aplicación del plugin de cambio de escena:



En la imagen se puede ver cómo cuando en el video ocurre un cambio de escena o una gran modificación en el video, este es detectado por el plugin y es indicado en la lista que se ve a la izquierda del video.

7 Conclusiones

Como evaluación final, haremos una retrospectiva para analizar el resultado del esfuerzo que se ha dedicado al proyecto y un balance de las metas trazadas y cumplidas. También, un análisis de cómo los caminos elegidos a lo largo del proceso influyeron en los resultados finales.

Las decisiones tomadas son siempre puntos discutibles, y en este caso se tomaron algunas que fueron de vital importancia para el desarrollo del proyecto. La primer decisión importante fue la del uso del lenguaje C++ para la implementación de la aplicación. Este camino demostró ser un arma de doble filo. Por un lado resultó ser un lenguaje bastante complicado de aprender, sin embargo demostró ser un lenguaje muy potente y que tiene la gran ventaja de contar con una cantidad de librerías, información y diversas herramientas al alcance del estudiante. A su vez, fue una gran ventaja a la hora de incorporar la librería del VAP al proyecto Skopeo, generando al fin de cuentas, un balance positivo a favor de la decisión. Es necesario comentar que los tiempos que fueron estimados a priori para que el grupo tuviera la capacidad de implementar los primeros bocetos de aplicación fueron más cortos de lo necesario, esto fue debido al largo periodo de investigación que exigió el proyecto.

Otra elección de gran importancia, fue la de utilizar la librería wxWidgets para el desarrollo de la interfaz gráfica. Por más que llevó más tiempo del que se pensaba el encontrar una librería que cumpliera con todos los requerimientos que se exigían (portabilidad, simpleza, buena documentación, etc.) esta librería demostró ser una herramienta muy potente y dúctil, adaptándose a necesidades para las que no había sido seleccionada en principio, como por ejemplo, el uso de ella en la implementación de la arquitectura de plugins.

La selección del tipo de comunicación que se iba a utilizar entre el VAP y la aplicación principal fue otro punto importante en el proyecto y también llevó su correspondiente tiempo de deliberación. Se cree que la decisión probó ser la correcta, dado que la comunicación cumplió y sobrepasó las expectativas de rendimiento en la transmisión de datos y demás utilidades usadas normalmente por la aplicación.

La arquitectura de plugins desarrollada cumplió con las expectativas que se tenían previamente. Provee de una forma clara y sencilla para que nuevos desarrolladores hagan sus propios plugins, los cuales extenderían las funcionalidades del programa enormemente. Cabe destacar que el acoplamiento de los plugins al programa anfitrión no acota en lo mas mínimo su versatilidad. Si bien el protocolo de comunicación es sencillo, los plugins tienen la posibilidad de ser completamente independientes del resto del sistema (pueden tener controles propios y acceder a los datos en forma independiente).

Se logro además proporcionar una interfaz gráfica amigable a las funcionalidades del VAP, lo cual constituía uno de los objetivos principales. En la actualidad, por intermedio de Skopeo, se puede no solo procesar y manipular información de video con todas las herramientas de que dispone el VAP, sino que también se podrá visualizar los resultados de este trabajo.

Todas estas metas alcanzadas, forman una base muy estable para poder lograr diversas metas en el fututo. En particular encontramos muy interesantes los siguientes aspectos a desarrollar; efectuar una prueba consistente de la portabilidad del sistema (garantizada debido a la utilización de una librería multiplataforma), la elaboración de plugins de audio y la incorporación de funcionalidades de audio a la aplicación (ver wxWave), la posibilidad de guardar los resultados en XML, la investigación y el posterior desarrollo de algoritmos mas complejoos, intentar efectuar un manejo de memoria más eficiente.

Nos parece de extrema importancia resaltar el aprendizaje efectuado a lo largo de proyecto. La investigación de nuevas disciplinas, el aprendizaje de nuevas herramientas, etc. Esta fue un experiencia muy enriquecedora no solo teniendo en cuenta los conocimientos inherentes al proyecto desarrollado sino como también por todo lo aprendido con referencia a la gestión del mismo.

Por lo tanto, entendemos que dicho balance es positivo dado que los objetivos principales han sido alcanzados, y porque además sentaron las bases para cumplir con los que han quedado pospuestos.

8 Bibliografía

- UML y Patrones, Craig Larman.
- WxWidgets Compared To Other Toolkits: http://www.wxwidgets.org/wiki/index.php/WxWidgets_Compared_To_Other_Toolkits
- FOX Community: Comparisons: http://www.fox-toolkit.net/cgi-bin/wiki.pl?Comparisons
- GUI-Toolkits http://bwachter.lart.info/documentation/guitoolkits.html
- Curso Taller de Verificación de software (INCO - FING): http://www.fing.edu.uy/inco/cursos/ingsoft/tvs/index.html
- wxWidgets Discussion Forum: http://wxforum.shadonet.com/
- Alphabetical class reference: http://www.wxwindows.org/manuals/2.6.3/wx_classref.html#classref
- wxWidgets homepage: http://wxwidgets.org/
- Plugin: http://en.wikipedia.org/wiki/Plugins
- Library http://en.wikipedia.org/wiki/Dynamic_linking
- Building a DLL with Visual C++ http://zone.ni.com/devzone/conceptd.nsf/webmain/3A1F4B7CBC68B636862567CC004D5CCA
- C Run-Time Libraries http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_c_run.2d.time_libraries.asp
- lf350, Software Development: Platform Independent Software Development http://www.tldp.org/linuxfocus/English/October2004/article350.shtml
- freshmeat.net: Category Reviews - GUI Toolkits for The X Window System http://freshmeat.net/articles/view/928
- RAD http://www.linuxfocus.org/Castellano/November2002/article265.shtml

9 Glosario

- **RAD:** Desarrollo Rápido de Aplicaciones (Rapid Application Development). Un sistema de programación que permite generar un programa rápidamente. Generalmente proveen de un número de herramientas para ayudar a desarrollar interfaces de usuario que normalmente llevarían un gran esfuerzo de desarrollo.

- **GNU LPGL:** La GNU Lesser General Public License (antes conocida como GNU Library General Public License) es una licencia de software libre publicada por la Fundación de Software Libre. Fué escrita en 1991 por Richard Stallman. La mayor diferencia entre GPL y LGPL es que la última puede ser vinculada a un programa no-LGPL (software libre o privado).

- **Buffer:** Ubicación de memoria en una computadora o en un instrumento digital reser-vada para el almacenamiento temporal de información digital, mientras que está espe-rando ser procesada.

- **Plugin:** Programa o aplicación que se acopla a otro mediante una interfaz.

- **IGU:** Interfaz gráfica de usuario.

10 Tabla de ilustraciones

11 Apéndice

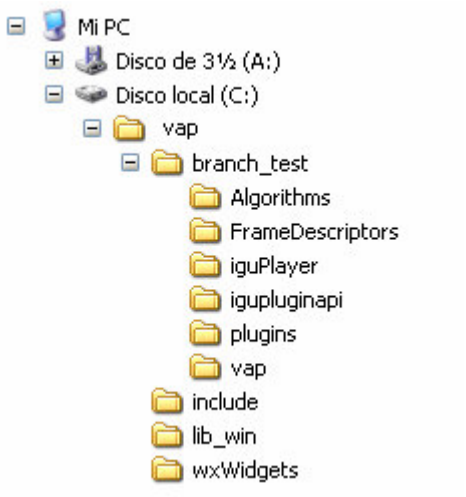
11.1 Instalación

11.1.1 Instalación de IguPlayer (para desarrollo)

Para instalar IguPlayer se debe realizar lo siguiente:

Copiar la carpeta IguPlayer en:	C:\vap\branch_test
Copiar la carpeta IguPluginApi en:	C:\vap\branch_test\igupluginapi
Copiar la carpeta Vap en:	C:\vap\branch_test\vap
Crear una carpeta Plugins[1] en:	C:\vap\branch_test

La estructura de carpetas debería quedar de la siguiente forma:



Abrir los siguientes proyectos con MVC++ y compilarlos.

IguPlayer.dsw
Vap.dsw

Luego de compilarlos deberíamos obtener lo siguiente:

IguPlayer.exe en:	C:\vap\branch_test\iguPlayer-3.2\Debug
Vap.lib en:	C:\vap\branch_test\vap\Debug

Luego de esto estamos listos para ejecutar IguPlayer.exe

[1] En este lugar se almacenarán los plugins creados. En el Apéndice C se detalla como crearlos.

11.1.2 Instalación de wxWidgets

Para instalar wxWidgets debemos hacer lo siguiente:

Ir a: `http://wxwidgets.org/downloads/`
Bajar la última versión disponible (wxWidgets 2.6.3).

Luego de bajarla (aprox. 20 Mb) descomprimirlo en una carpeta dentro de C:\vap y llamarla wxWidgets.
Abrir wxWindows.dsw ubicado en: C:\vap\wxWidgets\src\wxWindows.dsw y compilar todos los proyectos existentes.

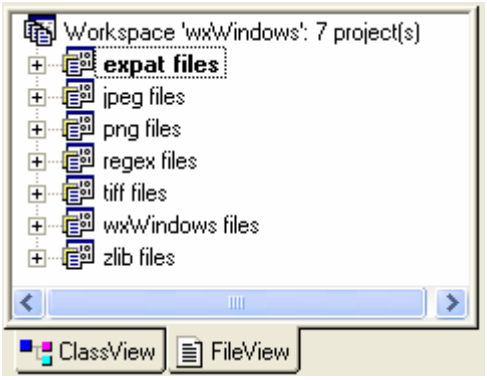


Figura 11-1

Notar que existe un error aquí. Seleccionar: Project Settings, Libary y en “Output file name” agregar “wx” al comienzo en cada librería (ver figura 10.1, 10.2 y 10.3) que se vaya a compilar. Por ejemplo: ..\..\lib\jpegd.lib debería ser: ..\..\lib\wxjpegd.lib

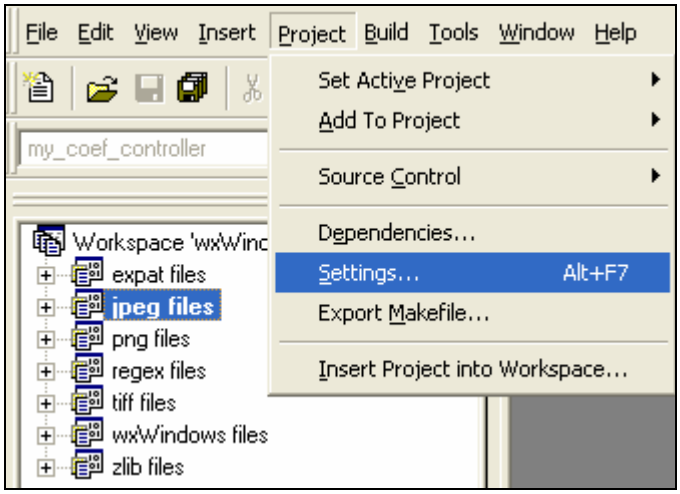


Figura 11-2

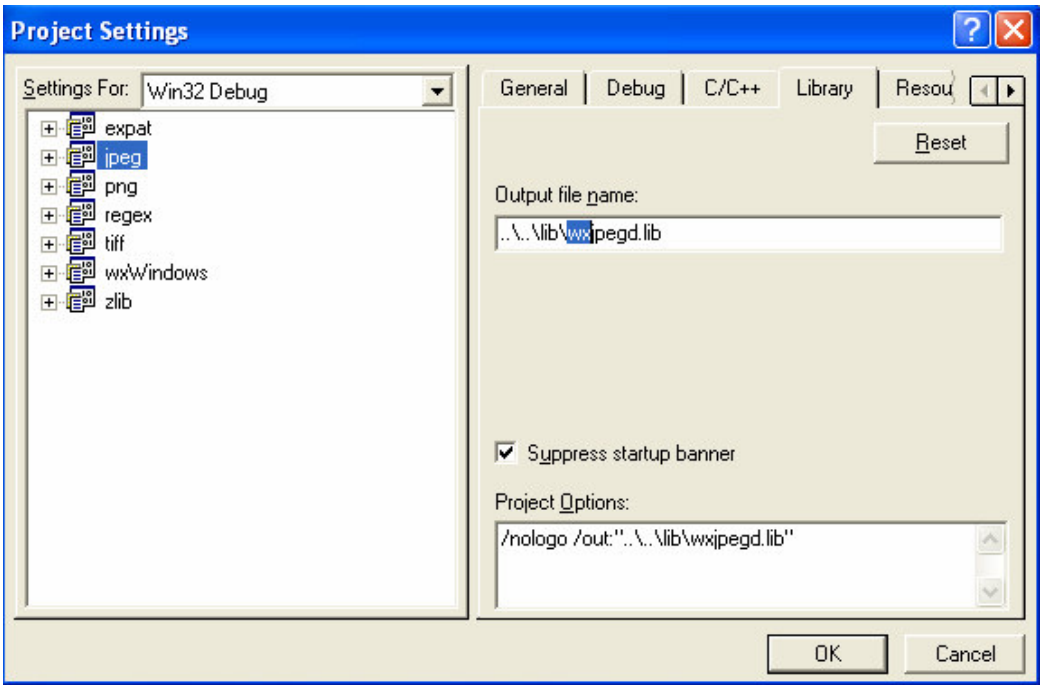


Figura 11-3

Adicionalmente se deben compilar otras librerías (para realizar aplicaciones más específicas), para ellos hay que abrir: C:\vap\wxWidgets\build\msw\wx.dsw.

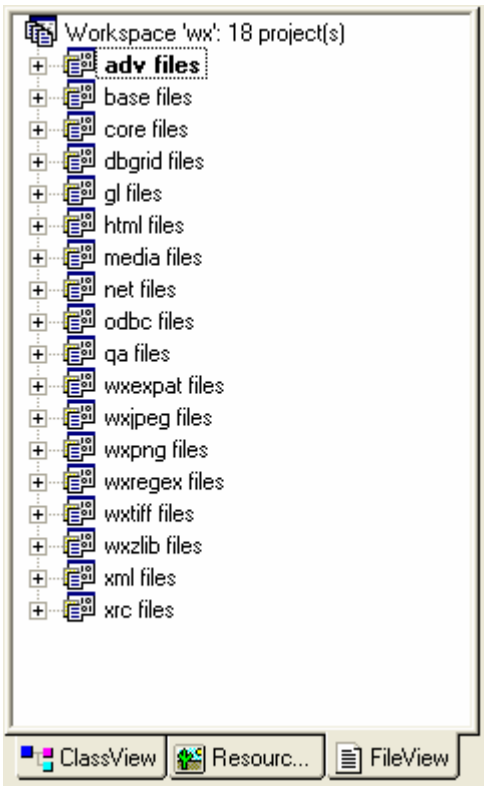


Figura 11-4

11.2 Manejo de IGU Player

11.2.1 Componentes básicos

Figura 11-5.- Controles básicos de video.

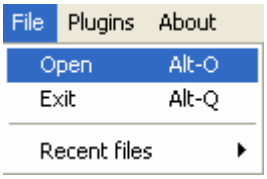


Figura 11-6.- Menú Archivo

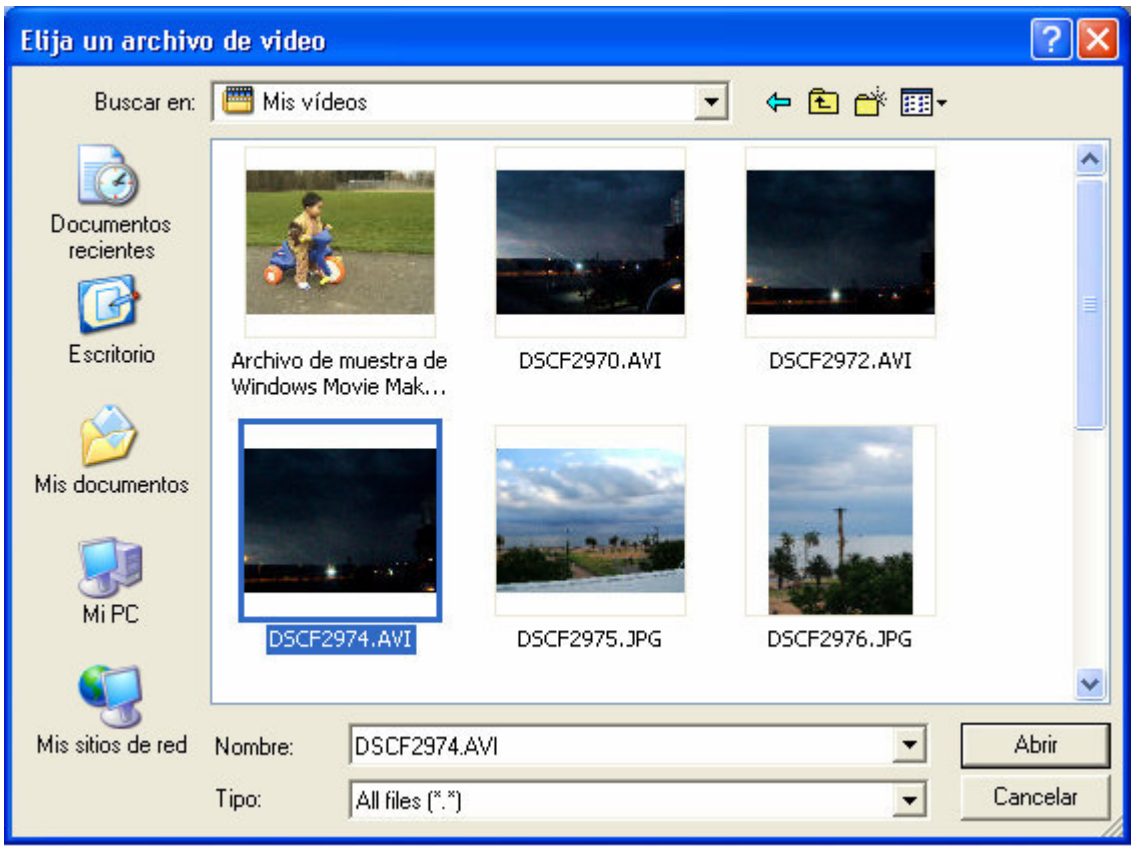


Figura 11-7.- Cuadro de diálogo de selección de archivo.

11.2.2 Plugins

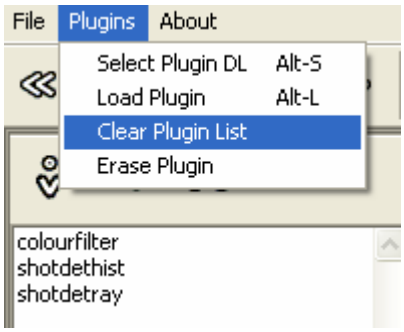


Figura 11-8.- Controles de plugin.

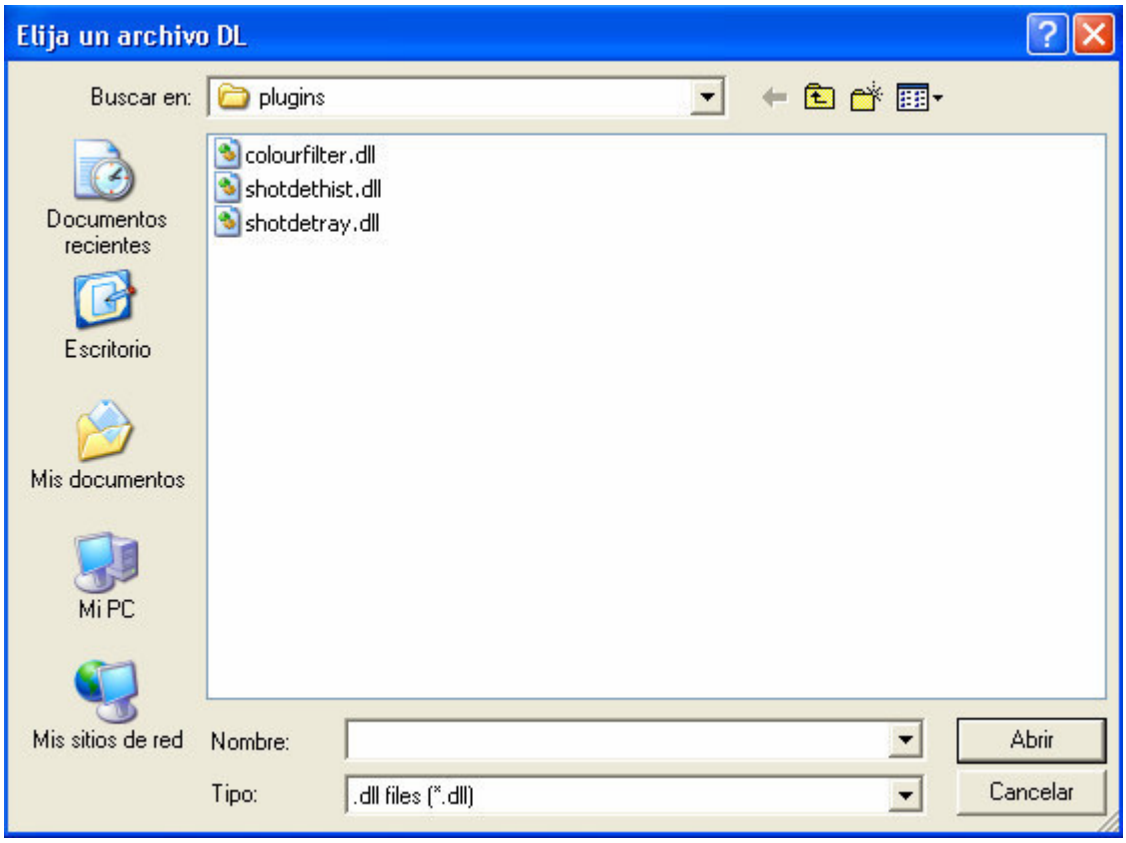


Figura 11-9.- cuadro de diálogo de elección de plugins.

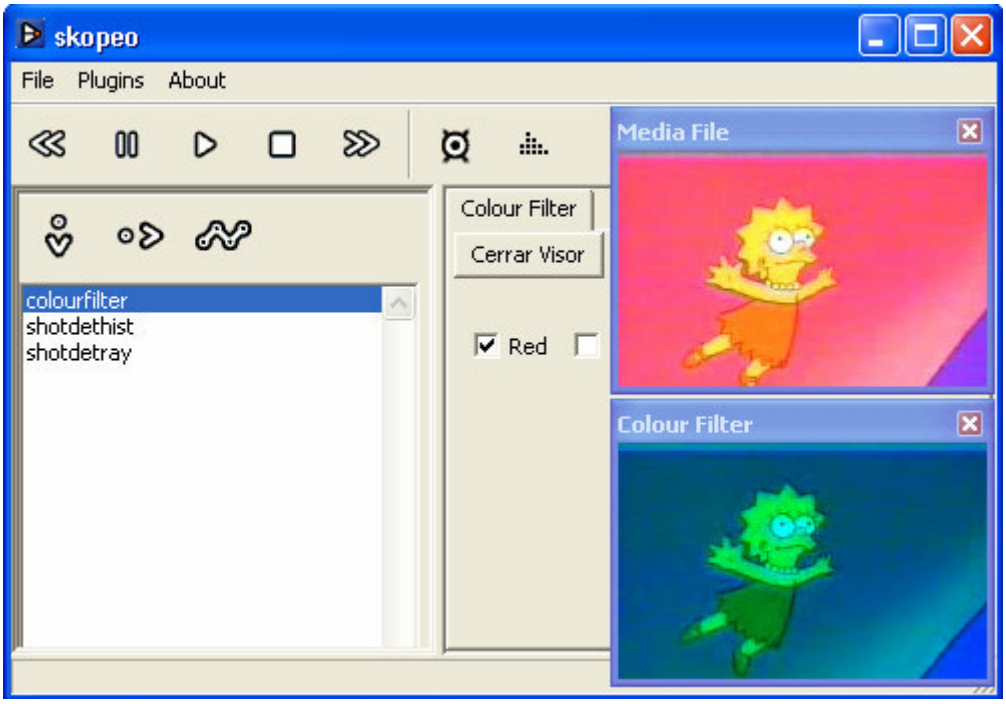


Figura 11-10.- Ejecución de un plugin (ColourFilter).

11.3 Pruebas y Testeos (pruebas pendientes)

Existen diversos tipos de pruebas que podríamos realizar a los efectos de verificar que el producto funcione bien, por ejemplo:

Unitaria - El propósito es probar módulos por separado para ver que éstos cumplan las funcionalidades definidas por su interfaz. Esto se hace creando programas auxiliares que se comunican con la interfaz.

Integración - El objetivo es encontrar defectos en las interfaces entre los diferentes módulos.

Funcional - El propósito aquí es detectar discrepancias entre la especificación funcional del sistema y su comportamiento real (se prueban los casos de uso).

Sistema - El objetivo es demostrar que el sistema cumple con sus requerimientos no funcionales. Existen diversos tipos: Volumen, carga/stress, seguridad, usabilidad, desempeño, configuración, instalación, documentación, etc.

Por ejemplo:
Volumen: Verifica el comportamiento frente a grandes cantidades de datos.
Carga: Verifica que el sistema funciona en forma aceptable cuando los recursos del sistema se saturan.
Desempeño: Verifica la respuesta y el tiempo de procesamiento.
Aceptación: Prueba previa a poner en producción el software. El objetivo es verificar que el software está listo y puede ser usado por el usuario final para realizar las funciones y tareas para las que fue construido.

Es este campo, de las pruebas, uno muy vasto dentro de la Ingeniería de Software al que podríamos dedicarle mucho tiempo y dedicación. Dada los cortos plazos que tenemos, nos vamos a centrar solamente en dos aspectos salientes: pruebas funcionales y de sistema.

11.3.1 Pruebas funcionales

Una buena forma de realizar estas pruebas es probar los casos de uso. En cada caso verificando si se cumple lo esperado.

Test #1	
Caso de uso	Reproduce video en tiempo real (play). (R1.1)
Datos de entrada	Video .mpg (18Mb)
Condiciones iniciales	Video cargado
Pasos de prueba	-
Resultados esperados	El video se reproduce en tiempo real sin errores
Resultados obtenidos	El video se pudo apreciar en timepo real sin errores

Test #2	
Caso de uso	Forward y Rewind (R1.4 y R1.5)
Datos de entrada	Video .mpg (18 Mb) y Plugin
Condiciones iniciales	Video reproduciendose
Pasos de prueba	-
Resultados esperados	El video debe avanzar y retroceder cuadro a cuadro
Resultados obtenidos	El video avanza perfectamente, el rewind no implementado debido a problemas con GetRGB(frame number)

Test #3	
Caso de uso	Carga Plugin
Datos de entrada	Video .mpg (18 Mb) y Plugin
Condiciones iniciales	Video cargado
Pasos de prueba	-
Resultados esperados	El plugin se carga en la lista
Resultados obtenidos	El plugin se carga sin problemas (con la salvedad de que admite plugins inválidos, el único requisito es que sea una .dll)

Test #4	
Caso de uso	Ejecuta plugin
Datos de entrada	Video .mpg (18 Mb) y Plugin
Condiciones iniciales	Video reproduciéndose.
Pasos de prueba	-
Resultados esperados	El plugin se ejecuta cuadro a cuadro
Resultados obtenidos	Se observa el resultado de la aplicación del plugin cuadro a cuadro.

Test #4	
Caso de uso	Concatena plugin
Datos de entrada	Video .mpg (18 Mb) y 2 Plugins
Condiciones iniciales	Video reproduciéndose
Pasos de prueba	-
Resultados esperados	Los plugins seleccionados se ejecutan sobre el video uno a continuación del otro, obteniendo el resultado de ejecutar un plugin tras otro.
Resultados obtenidos	Se observa el resultado de la aplicación de los dos plugins seleccionados.

11.3.2 Pruebas de Sistema

Test #5 - Volumen - Pentium IV/ 1Gb RAM	
Volumen de entrada	Video .mpg (600 Mb)
Tiempo reproducción	No se aprecian diferencias con un archivo de 20 Mb
Tiempo carga plugin	No se aprecian diferencias con un archivo de 20 Mb
Tiempo ejecución plugin	No se aprecian diferencias con un archivo de 20 Mb
Resultados esperados	Es de esperar que se presenten problemas.
Resultados obtenidos	El video se exhibe bien los primeros 60 segundos, luego se enlentece un poco, esto muestra que se debe mejorar el manejo de memoria.

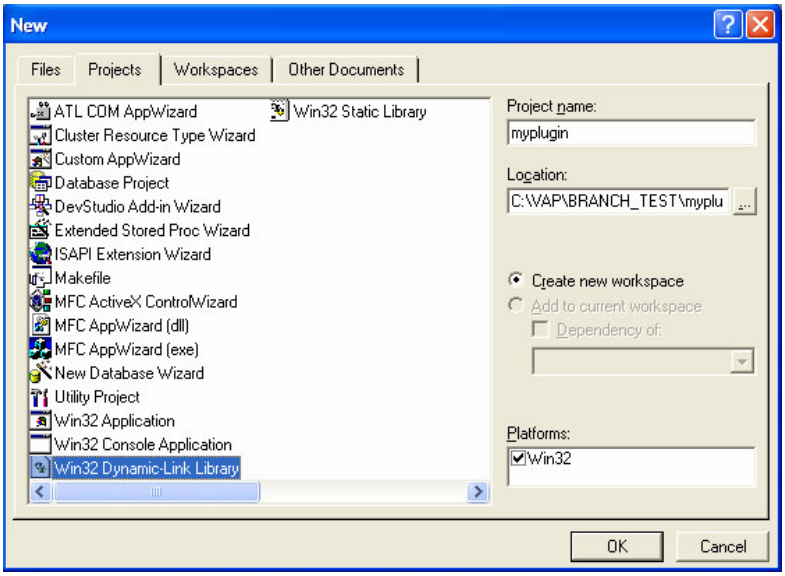
Test #6 - Carga - Video 20 Mb - PC Pentium III - 680 Mb RAM	
Recursos del sistema	Word, Winamp, Media Player, MVC++, Adobe Acrobat, Firefox: funcionando.
Tiempo reproducción	No se aprecian diferencias a cuando el sistema no estaba tan exigido
Tiempo carga plugin	No se aprecian diferencias a cuando el sistema no estaba tan exigido
Tiempo ejecución plugin	No se aprecian diferencias a cuando el sistema no estaba tan exigido
Resultados esperados	Es de esperar que se presenten problemas.
Resultados obtenidos	Concluimos que la exigencia no fue los suficiente como para observar problemas de enlenteciemiento (dado que los presenta con archivos mas grandes).

11.4 Desarrollo de Plugins

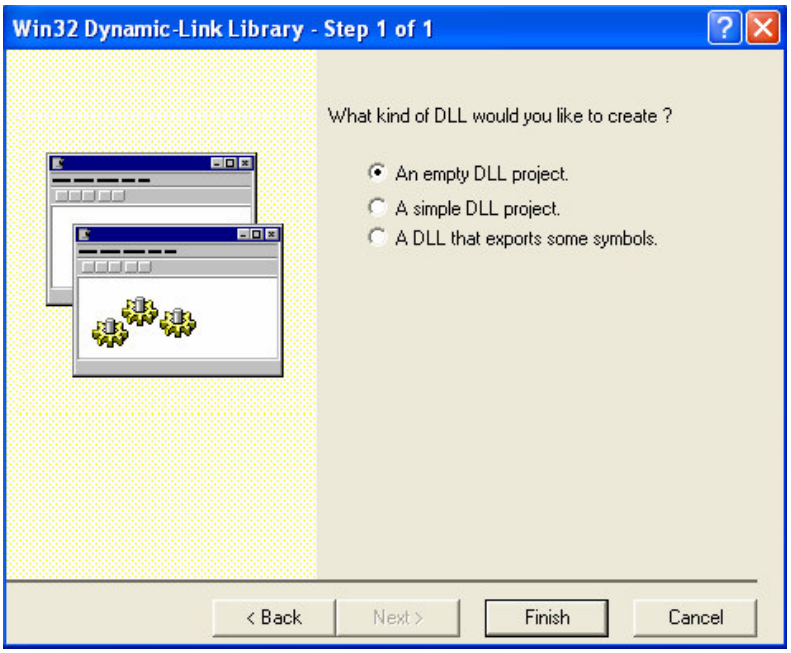
11.4.1 Seteos previos

Configuración del entorno de Desarrollo:

Ir a la ventana de creación de nuevos proyectos en MVS6.0 y elegir la opción Win 32 Dynamic Link Library. Darle el nombre del plugin, sin espacios ni guiones. Por ejemplo myplugin. Ubicar la carpeta de dicho proyecto en la carpeta vap/branch_test.



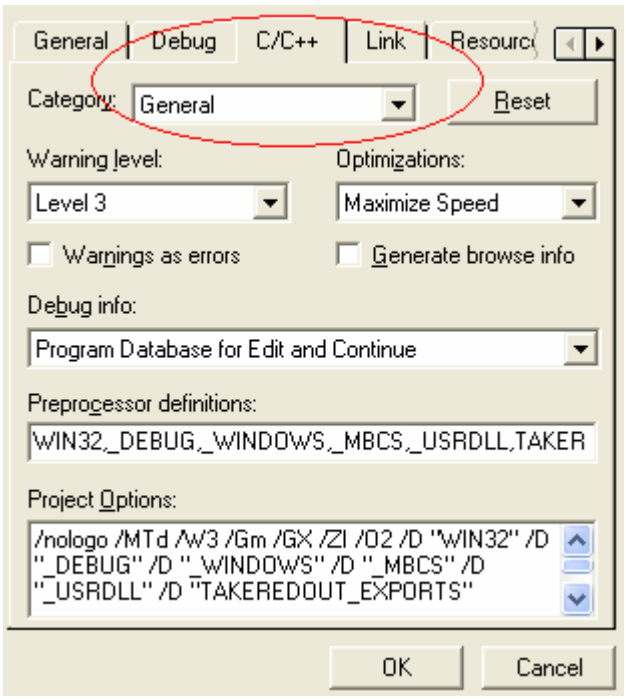
Elegir la opción de crear un proyecto vacío de DLL: “An empty DLL Project”



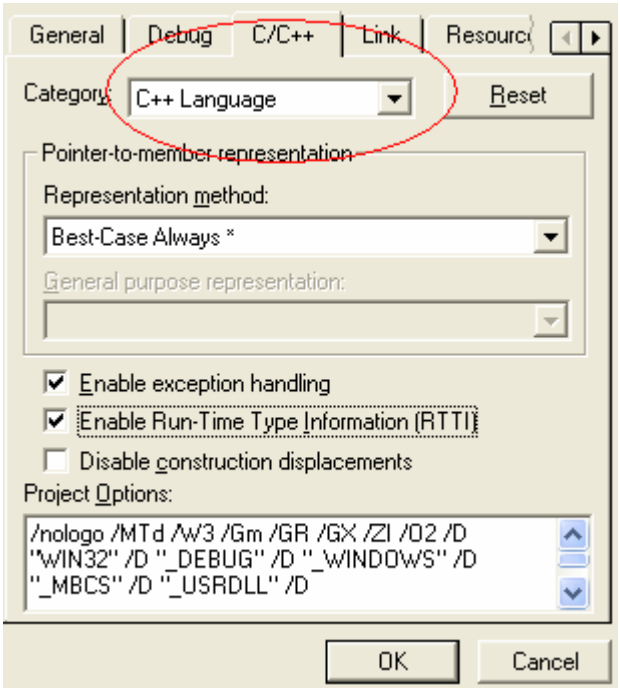
Configuración del proyecto:

Project → Settings → C/C++:

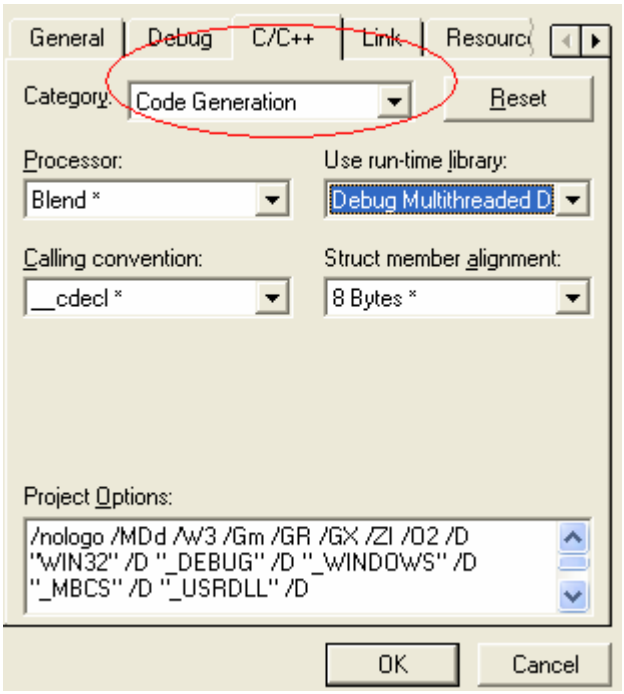
En la sección **General**, elegir **Maximize Speed** en la combolist **Optimizations**, en **Debug Info** elegir **Program Database**. (Dejar **Warning Level** sin modificar).



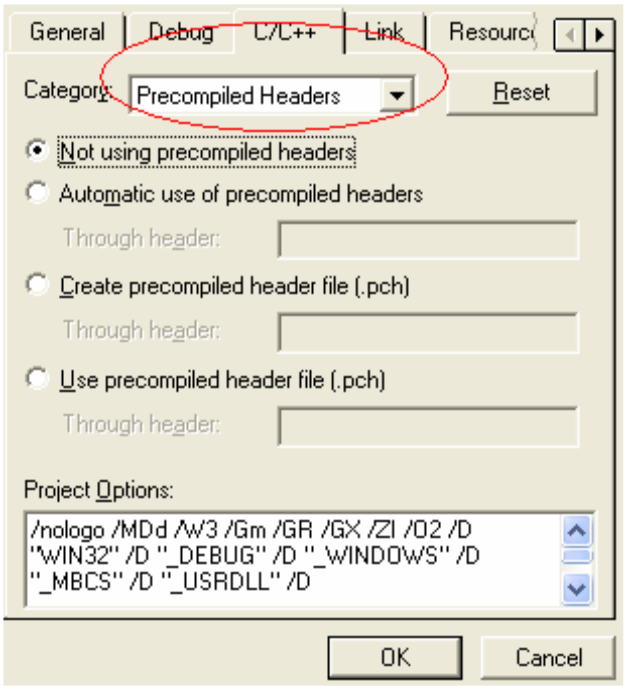
En la sección C++ Language seleccionar Enable exception handling y Enable Run Time Information.



En **Code Generation** elegir **Debug Multithreaded DLL** como **Use Run Time Library**. Dejar las otras opciones sin modificar.



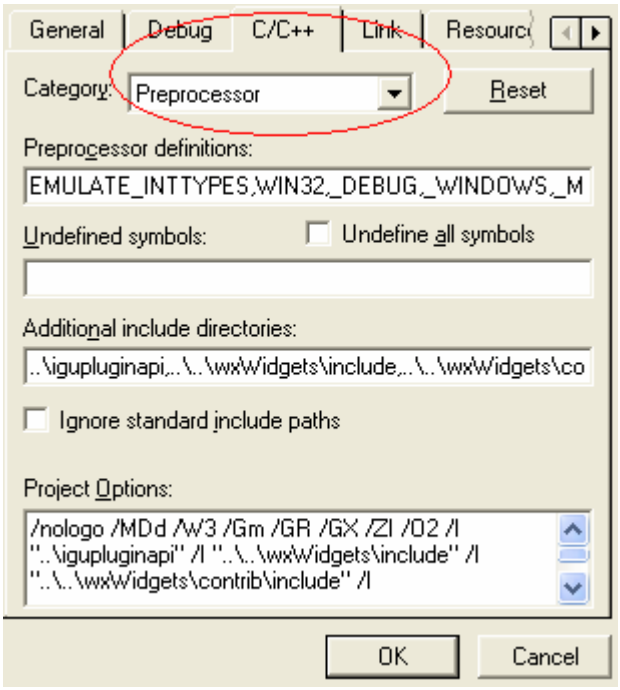
Ir a Precompiled Headers y elegir “Not using precompiled headers”.



En **Preprocessor→ Preprocessor Definitions** agregar EMULATE_INTTYPES a las definiciones que ya están por defecto.

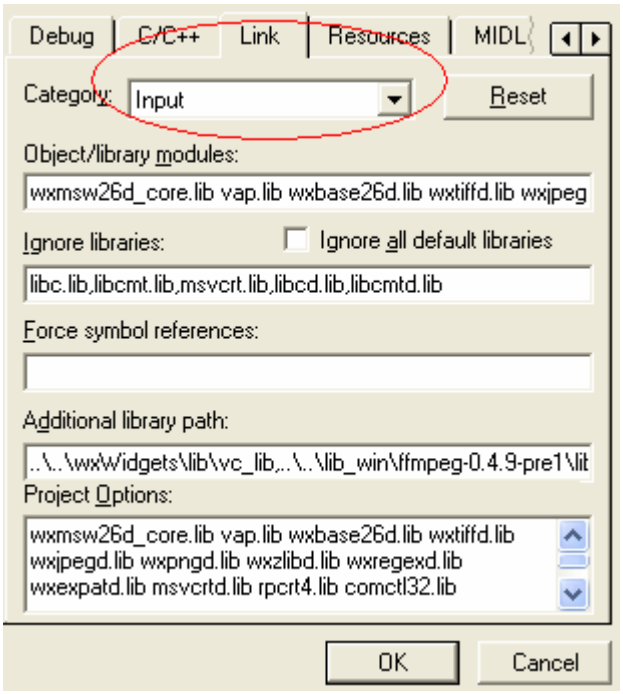
En Additional Include Directories agregar:

..\igupluginapi,..\..\wxWidgets\include,..\..\wxWidgets\contrib\include,..\..\wxWidgets\lib\vc_lib\mswd,..\..\include\fobs,..\..\include\ffmpeg\libavformat,..\..\include\ffmpeg,..\..\include\InsightToolkit-2.0.1\Code\Common,..\..\lib_win\VC6\InsightToolkit-2.0.1-bin,..\..\include\InsightToolkit-2.0.1\Utilities\vxl\vcl,..\..\include\InsightToolkit-2.0.1\Utilities\vxl\core,..\..\lib_win\VC6\InsightToolkit-2.0.1-bin\Utilities\vxl\core,..\..\lib_win\VC6\InsightToolkit-2.0.1-bin\utilities\vxl\vcl,..\..\include\InsightToolkit-2.0.1\Code\BasicFilters,..\..\include\InsightToolkit-2.0.1\Code\IO,..\..\include\InsightToolkit-2.0.1\bin\Utilities\,..\..\include\InsightToolkit-2.0.1\Code\Numerics\Statistics,..\..\lib_win\VC6\InsightToolkit-2.0.1-bin\Utilities,..\..\include,..\Algorithms,..\FrameDescriptors,..\vapCore,..\..\include\ffmpeg\libavcodec



Project → Settings → Link

En la Categoría **Input**, en el campo **Object/library** modules incluir las siguientes librerías además de las que ya se encuentran por defecto:
wxmsw26d_core.lib vap.lib wxbase26d.lib wxtiffd.lib wxjpegd.lib wxpngd.lib wxzlibd.lib wxregexd.lib wxexpatd.lib msvcrtd.lib rpcrt4.lib comctl32.lib strmiids.lib ITKNrrdIO.lib ITKCommon.lib ITKBasicFilters.lib ITKIO.lib ITKCommon.lib itkvnl_inst.lib itkvnl_algo.lib itkvnl.lib itkvcl.lib itknetlib.lib itksys.lib itkpng.lib itktiff.lib itkzlib.lib itkjpeg8.lib itkjpeg12.lib itkjpeg16.lib ITKMetaIO.lib ITKDICOMParser.lib ITKEXPAT.lib avformat.lib avcodec.lib xerces-c_2D.lib
En el campo **Ignore libraries**, incluir estas librerías: libc.lib, libcmt.lib, msvcrt.lib, libcd.lib, libcmtd.lib
En **Additional Library path**, incluir estos directorios: ..\..\wxWidgets\lib\vc_lib,..\..\lib_win\ffmpeg-0.4.9-pre1\libavcodec,..\..\lib_win\ffmpeg-0.4.9-pre1\libavformat,..\..\lib_win\VC6\InsightToolkit-2.0.1-bin\bin\Debug,..\..\lib_win\xerces\lib,..\..\lib_win\xerces\bin,..\vap\Debug



11.4.2 Como desarrollar e integrar plugins

Para desarrollar un nuevo plugin, se deben escribir 2 archivos, un ".h" y un ".cpp".
Por ejemplo myplugin.h y su respectivo myplugin.cpp.

A continuación se explica lo que se debe agregar para poder empezar a trabajar en la implementación de algoritmos:

myplugin.h

```
// Definiciones y directivas de preprocesador
#ifndef _MYPLUGIN_H_
#define _MYPLUGIN_H_

#if defined(__GNUG__) && !defined(NO_GCC_PRAGMA)
#pragma interface "myplugin.h"
#endif

#include "igupluginapi.h"                    // inclusión de la interfaz

// También se pueden incluir cualquier librería de wxWidgets o del vap que se
// Necesiten  específicamente para el algoritmo de interés y que no se
// Encuentren en el api.

class myPlugin: public iguPluginapi          // la clase de nuestro plugin debe
// heredar del api
{

// Aquí se deben declarar los métodos que se heredaron y por supuesto, los
// nuevos métodos que nuestro plugin necesite, junto con todos los atributos
// necesarios (elementos del VAP o controles de wxWidgets).

}

#endif
```

myplugin.cpp

```
// definiciones y directivas de preprocesador:
#if defined(__GNUG__) && !defined(NO_GCC_PRAGMA)
#pragma implementation "myplugin.h"
#endif

#include "wx/wxprec.h"

#ifdef __BORLANDC__
#pragma hdrstop
#endif

#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif

#include "myplugin.h"

IMPLEMENT_IGUPLUGINAPI(myPlugin)       // Macro que implementa el
// Punto de entrada de la dll, y la función que es exportada para crear el objeto
// del plugin. Aquí se implementan los métodos que declaramos en myplugin.h
```

Ejecución del plugin:

Luego de compilar el proyecto, hay dos maneras de cargar la dll obtenida. La primera es colocarla en la carpeta vap/branch_test/plugins, y el host la cargara automáticamente al inicializarse. Esta forma es más apropiada para cuando la dll ya ha sido probada.

La segunda manera es cargarla uno mismo, y esto se logra realizando lo siguiente:

Abrir el iguPlayer.
Seleccionar un archivo de video: Ir a File → Open y navegar hasta seleccionar un archivo de video.
Cargar la dll: Ir a Plugins → Load Plugin.
Si la dll fue escrita correctamente en el panel de la izquierda aparecerá la sección preparada para el mismo.

11.5 WxWidgets

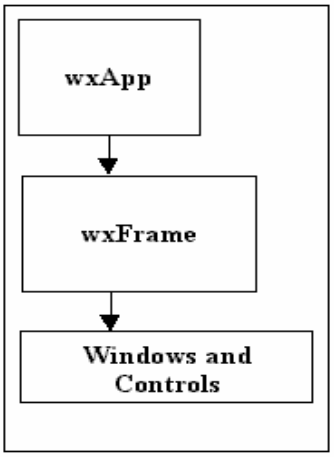
11.5.1 ¿Qué es wxWidgets?

Inicialmente comenzó como un proyecto para crear aplicaciones portables entre Unix y Windows (creada por Julian Smart en la Universidad de Edinburgo, 1992) pero con el tiempo se ha ido extendiendo a Mac, WinCE y otras tantas plataformas.
wxWidgets posee la un sencillo API para escribir aplicaciones de interfaz gráfica y una muy buena ayuda (material, foros[2] de discusión, etc.).

11.5.2 Requerimientos básicos

En Windows es necesario tener un compilador soportado por wxWidgets (MS Visual C++ o Borland C++).
En Linux es necesario tener una versión de GCC (compilador Open Source para GNU) instalado.
Dado que wxWidgets soporta tantas plataformas diferentes y compiladores es bastante impráctico proveer las librerías precompiladas, es por eso que cada uno debe compilarlas por si mismo.

11.5.3 Cómo funciona



Antes que nada debemos agregar: **#include "wx/wx.h"** (encabezados de wxWidgets).
Un programa consiste básicamente en una clase derivada de **wxApp**. Sobrescribiendo su método **OnInit()** se inicializa el programa.

```
class MyApp: public wxApp{virtual bool OnInit();}
```

La ventana principal se crea derivando de la clase **wxFrame** y luego podemos agregar un menú y una barra de estado en su constructor.

```
class MyFrame: public wxFrame{ public: MyFrame(const wxString& title, const wxPoint& pos, const wxSize& size); }
```

Para que la aplicación responda a eventos es necesario declarar una **tabla de eventos** usando el siguiente macro:

```
DECLARE_EVENT_TABLE()
```

Es necesario pues usar manipuladores de eventos (handlers).

```
void OnAbout (wxCommandEvent& event);
void OnQuit(wxCommandEvent& event);
```

Además, para responder a un comando de menú debe de tener un identificador único:

```
enum {
ID_Quit = 1;
ID_About = 2;
}
```

[2] http://wxforum.shadonet.com

Luego hay que implementar la tabla de eventos en la que los eventos son ruteados a los respectivos manipuladores de eventos (para los que hay macros predefinidos para eventos comunes).

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
EVT_MENU(ID_Quit, MyFrame::OnQuit)
EVT_MENU(ID_About, MyFrame::OnAbout)
END_EVENT_TABLE()
```

En wxWidgets la función “main” es implementada en el siguiente macro, que crea una instancia de la aplicación y arranca el programa:

```
IMPLEMENT_APP(MyApp)
```

Veamos pues como crear un menú:

```
wxMenu *menuFile = new wxMenu;
menuFile->Append( ID_About, "&About..." );
menuFile->AppendSeparator();
menuFile->Append( ID_Quit, "E&xit" );

wxMenuBar *menuBar = new wxMenuBar;
menuBar->Append( menuFile, "&File" );
SetMenuBar( menuBar );
CreateStatusBar();
SetStatusText( "Welcome to wxWidgets!" );
}
```

Y los manipuladores de eventos:

```
void MyFrame::OnQuit(wxCommandEvent& WXUNUSED(event)){
Close(true);
}

void MyFrame::OnAbout(wxCommandEvent& WXUNUSED(event)){
wxMessageBox("Hello world",
"About Hello World", wxOK | wxICON_INFORMATION, this);
}
```

Veamos que obtenemos con el código referido arriba:

11.5.4 Dibujo de imágenes con wxWidgets

Las herramientas de wxWidgets que se usan para el dibujo de imágenes en pantalla son tres, las clases wxImage, wxClientDC y wxBitmap.
El proceso se lleva a cabo de la siguiente manera y en el siguiente orden:
Se crea un objeto wxClientDC para poder pintar en el área de una ventana.
Luego se crea un objeto wxImage para manejar la imagen que se va a desplegar.
Se genera un objeto wxBitmap basado en el objeto wxImage creado en el paso anterior.
Se usa el método DrawBitmap del ClientDC para mostrar la imagen en la ventana seleccionada.

- WxClientDC permite pintar en el área de la ventana que está predestinada para eso, esta clase maneja todos los eventos de pintura en la ventana.
- WxImage es una clase que encapsula una imagen independiente de la plataforma, se puede cargar desde un archivo o se puede generar desde datos. Una imagen cargada con wxImage no puede ser mostrada directamente con un ClientDC, por lo que es necesario generar un bitmap dependiente de la plataforma que será el que se mostrará en la ventana.
- WxBitmap encapsula un bitmap dependiente de la plataforma en la que se está trabajando. Se usa para generar un bitmap que permita mostrar el objeto imagen wxImage.