# INFRAESTRUCTURA MULTIPLATAFORMA PARA EL DESARROLLO DE APLICACIONES DE REALIDAD VIRTUAL.

**DANIEL FELIPE MEJIA PADILLA**

**UNIVERSIDAD DE LOS ANDES**
**FACULTAD DE INGENIERÍA**
**MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**
**BOGOTÁ, COLOMBIA**
**2004**

# INFRAESTRUCTURA MULTIPLATAFORMA PARA EL DESARROLLO DE APLICACIONES DE REALIDAD VIRTUAL.

**DANIEL FELIPE MEJIA PADILLA**

**Tesis de grado para optar al título de:**
**Magíster en Ingeniería de Sistemas y Computación**
**Asesores:**
**Dr. Pablo Figueroa**
**Dr. José Tiberio Hernández**

**UNIVERSIDAD DE LOS ANDES**
**FACULTAD DE INGENIERÍA**
**MAGISTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**
**BOGOTÁ, COLOMBIA**
**2004**

Dedicado a mis padres,
mi hermano e Isabel
.

## AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

Pablo Figueroa, asesor principal de la tesis, por todo su apoyo, orientación y sus valiosos aportes a lo largo de todo el proyecto.

Marcela Hernández, jurado de tesis, por su tiempo, participación y colaboración en el desarrollo de los ejemplos.

Silvia Takahashi, coordinadora de la Maestría, por la ayuda que me brindo durante todo el cursar de mi Maestría.

Fernando de la Rosa, por su apoyo y colaboración para ayudarme a establecer la idea original del proyecto.

Andrea Herrera, por el apoyo y la colaboración que me brindo durante todo el desarrollo del proyecto.

Finalmente, al Departamento de Ingeniería de Sistemas y Computación, por la oportunidad brindada para la realización de la Maestría.

# CONTENIDO


1. INTRODUCCIÓN ..... 10

1.1. Objetivos Generales ..... 11
1.2. Objetivos Específicos ..... 11
1.3. Justificación ..... 12

2. ESTADO DEL ARTE DE LAS HERRAMIENTAS DE DESARROLLO DE SOFTWARE DE RV ..... 14

2.1. Herramientas de Software ..... 14
2.1.1. SYSYGY ..... 14
2.1.2. DICELIB: Distributed Cave Engine Library ..... 16
2.1.3. VrJuggler ..... 17
2.1.4. Performer ..... 18
2.1.5. CAVELIB ..... 18
2.1.6. VRPN: Virtual-Reality Peripheral Network ..... 19
2.1.7. Alice ..... 19
2.1.8. WTK: World Toolkit ..... 20
2.2. Plataformas ..... 22
2.2.1. ARSBOX ..... 22
2.2.2. Sara Virtual Reality facilities ..... 22
2.2.3. VISBOX ..... 23
2.2.4. SGI Reality Center ..... 23
2.2.5. FAKESPACE ..... 24
2.2.6. BARCO ..... 24
2.2.7. MULTIGEN ..... 24
2.3. Metodologías de desarrollo para Realidad Virtual ..... 25
2.3.1. MARYGOLD ..... 25
2.3.2. PMIW ..... 25
2.4. Arquitecturas de desarrollo para Realidad Virtual ..... 26
2.4.1. MVC & PAC ..... 26
2.4.2. MARYGOLD Toolset ..... 28
2.4.3. InTml: Interaction Techniques Markup Language ..... 28

2.5. Requerimientos ..... 29

3. DEFINICIÓN DE LA PROPUESTA DEL AMBIENTE DE DESARROLLO ..... 31

3.1. Hardware ..... 32
3.1.1. Servidor de dispositivos ..... 33
3.1.2. HMD ..... 33
3.1.3. Fish Tank Desktop ..... 34
3.1.4. Stereo Pc ..... 36
3.1.5. PC ..... 36
3.1.6. GEOWALL ..... 36
3.2. Software ..... 37
3.2.1. Integración de Librerías ..... 47
3.3. Procedimientos ..... 52
3.3.1. Configuración (Conexión HW y Calibración) ..... 52
3.3.2. Desarrollo ..... 55
3.3.3. Demos ..... 58

4. EJEMPLOS ..... 67

4.1. Visualizador Performer ..... 67
4.2. Visualizador Performer con Posición de Cabeza ..... 78
4.3. Visualizador DCom ..... 80

5. CONCLUSIONES Y TRABAJO FUTURO ..... 87

6. BIBLIOGRAFÍA ..... 90

APÉNDICES ..... 96

Pasos para configurar un ambiente fish tank en Linux ..... 96
Archivo de Configuración XFREE86 ..... 97
Árbol de Directorios CD Anexo ..... 100
Arquitectura InTml ..... 103

# LISTA DE IMÁGENES

Imagen 1 Plataforma de SYSYGY [SYS 03] ..... 15
Imagen 2 Arquitectura de VrJuggler [VRJ 03] ..... 17
Imagen 3 Configuración del Laboratorio ..... 32
Imagen 4 Fish Tank Desktop ..... 35
Imagen 5 Diagrama de Capas de Configuración del Software ..... 37
Imagen 6 Infraestructura de Software ..... 39
Imagen 7 Diagrama de Clases Framework InTml (Información) ..... 40
Imagen 8 Diagrama de Clases Framework InTml (Puertos) ..... 42
Imagen 9 Diagrama de Clases Framework InTml (Filtros) ..... 43
Imagen 10 Diagrama de Clases Framework InTml (Sistema) ..... 44
Imagen 11 Diagrama de Clases Framework Intml (VRJUGGLER + PERFORMER) ..... 45
Imagen 12 Diagrama de Clases Framework Intml (Ambiente de Ejecución) ..... 46
Imagen 13 Instalación de los Dispositivos ..... 53
Imagen 14 Mesa para calibración de los Trackers ..... 54
Imagen 15 Descripción de un nuevo filtro de Interacción (Key2RGB) ..... 57
Imagen 16 Descripción de un nuevo filtro de dispositivos ..... 57
Imagen 17 Descripción de un nuevo filtro de visualización ..... 58
Imagen 18 Filtro de Visualización Cube ..... 60
Imagen 19 Filtro para un dispositivo ..... 60
Imagen 20 Filtro de Interacción ..... 61
Imagen 21 Aplicación Ejemplo ..... 61
Imagen 22 Aplicación de Ejemplo Modificada con interacción 1 ..... 63
Imagen 23 Cambio en Filtro de Interacción ..... 64
Imagen 24 Aplicación de Ejemplo Modificada con interacción 2 ..... 64
Imagen 25 Filtro representativo de un Gamepad ..... 65
Imagen 26 Aplicación de Ejemplo Modificada para el Gamepad ..... 65
Imagen 27 Imágenes de la Aplicación Visualizador Performer ..... 67
Imagen 28 Visualizador de Objetos Performer ..... 68
Imagen 29 Diagrama de Clases Filtro FobTracker ..... 68
Imagen 30 Diagrama de Clases Filtro IsInZone ..... 70
Imagen 31 Comportamiento basado en proyección de vectores ..... 72
Imagen 32 Diagrama de Clases Filtro OrbitZoom ..... 72
Imagen 33 Diagrama de Clases Filtro PerformerFile ..... 74
Imagen 34 Imágenes de la Aplicación en Ambiente Fish Tank Desktop ..... 78
Imagen 35 Visualizador de Objetos Performer en un Fish Tank ..... 79
Imagen 36 Imágenes de la Aplicación Visualizador DCom ..... 80
Imagen 37 Visualizador de Imágenes DCOM ..... 81
Imagen 38 Diagrama de Clases Filtro Slide3D ..... 82
Imagen 39 Diagrama de Clases Filtro SlideObject ..... 82

Imagen 40 Diagrama de Clases Filtro VTKDComImage ..... 83
Imagen 41 Ejemplo de un filtro (SelectByTouching) [FIG 04] ..... 103
Imagen 42 Modelo de Ejecución de INTML [FIG 04] ..... 104
Imagen 43 Entidades y Relaciones en INTML [FIG 04] ..... 106
Imagen 44 Arquitectura de una implementación INTML [FIG 02] ..... 107

# LISTA DE TABLAS

Tabla 1 Comparación de las Herramientas ..... 21
Tabla 2 Método main de la Clase MainInTmlLoader ..... 46
Tabla 3 Método SetApp de la Clase VRJPFApp ..... 48
Tabla 4 Archivo XML de Configuración para VrJuggler habilitando VRPN ..... 48
Tabla 5 Archivo XML de Configuración para Habilitar el administrador de Entradas ..... 49
Tabla 6 Ejemplo de VTK mace.cxx ..... 51
Tabla 7 Uso de VtkActorToPF ..... 51
Tabla 8 Configuración de Dispositivos ..... 53
Tabla 9 Archivo InTml ..... 62
Tabla 10 Creación de los Puertos en el Filtro FobTracker ..... 69
Tabla 11 Método Execute de FobTracker ..... 69
Tabla 12 Metodo CreatePorts de IsInZone ..... 70
Tabla 13 Metodo execute de IsInZone ..... 70
Tabla 14 Método createPorts de OrbitZoom ..... 73
Tabla 15 Metodo execute de OrbitZoom ..... 73
Tabla 16 Método createPorts de PerformerFile ..... 74
Tabla 17 Metodo execute de PerformerFile ..... 75
Tabla 18 Código de la Aplicación Generic Performer Viewer ..... 76

# 1. INTRODUCCIÓN

Un objetivo importante desde los inicios de la computación gráfica ha sido lograr imitar la realidad por medio de modelos gráficos y de simulación. Con la propuesta de Sutherland (Espada de Damocles)[1] se abrió un nuevo concepto, ya no sólo se quiere imitar la realidad sino que se quiere engañar a la mente, haciéndole creer que el usuario esta en el mundo y puede interactuar con él.

A pesar de la simplicidad de la propuesta original de Sutherland, lograba que la persona se compenetrara con el mundo virtual de una manera aceptable, sin embargo ésta tenia una gran desventaja, la construcción de toda una infraestructura mecánica para sostener el dispositivo de visualización. Esta construcción no sólo tenía un costo muy elevado sino que además limitaba la interacción del usuario. Estas limitantes han ido desapareciendo con la evolución de la tecnología, debido a la aparición de sistemas de visualización y dispositivos de interacción como cascos y guantes, que permiten crear ambientes mucho más complejos y fáciles de usar que el planteado por Sutherland.

En la actualidad existen diferentes opciones de software y de hardware que permiten crear ambientes de Realidad Virtual (RV), particularmente los ambientes de inmersión. Ya no sólo se consiguen los dispositivos independientemente del software sino que existen opciones en el mercado de ambientes completos y empresas dedicadas especialmente a la creación de éstos, pero con costos aun muy elevados.

Todo este conjunto de elementos (hardware y software), busca facilitar el desarrollo de aplicaciones de RV, dándole herramientas a los desarrolladores para la elaboración de una

[1] Primer casco visor de RV, constaba de tubos de rayos catódicos en un armazón de alambre [PER 03]

nueva aplicación. Se busca transformar el desarrollo en un proceso de adaptación, el cual se logra a través de la interacción de los elementos de la infraestructura. En la actualidad, todos estos elementos requieren de un conocimiento altamente técnico y no permiten el desarrollo de una aplicación rápidamente, dado que se debe cumplir con la curva de aprendizaje de las herramientas.

Es por esto que el objetivo de este trabajo es generar una infraestructura que permita, interactuar con la mayor cantidad de tecnología de RV, realizar desarrollos de software en un menor tiempo y disminuir los costos de montaje del ambiente de desarrollo.

## 1.1. Objetivos Generales

Los Objetivos Generales de este Trabajo son los siguientes:

- Generar un plataforma[2] de trabajo lo suficientemente completa que permita el desarrollo de aplicaciones de RV.
- Evaluar la plataforma por medio del desarrollo de demos útiles para la investigación y el grupo de Informática gráfica de departamento.

## 1.2. Objetivos Específicos

- Analizar el estado actual de los elementos de software, hardware, metodologías y procedimientos para el desarrollo de aplicaciones de RV.
- Sintetizar la información recopilada para escoger los elementos más útiles para la generación de la plataforma de software de la infraestructura planteada.
- Generar una primera versión de la infraestructura a ser utilizada en el laboratorio de computación gráfica y realidad virtual de la Universidad de los Andes.

[2] Conjunto de elementos de hardware y software unidos en una misma infraestructura

- Establecer los procedimientos necesarios para el uso efectivo de la infraestructura montada.
- Desarrollar dos demos particulares sobre la infraestructura montada, para la reconstrucción de imágenes médicas y la visualización de objetos gráficos en Performer [SGI 04], motivados en las necesidades del grupo de informática gráfica..

## 1.3. Justificación

En la actualidad existen diferentes áreas que han demostrado la importancia de la tecnología de RV y las ventajas que está puede brindar. Como por ejemplo las excavaciones petroleras, disminuyendo los riesgos en sus perforaciones, por medio de la toma de decisiones mas certeras y efectivas, el diseño CAD[3] buscando una mejor interacción con los modelos y el entretenimiento con nuevas opciones de diversión, evidenciando que esta tecnología que pareció desvanecerse en algún momento ofrece un apoyo útil en cada una de estás.

Es por esto que la Universidad de los Andes está retomando sus investigaciones en el área de Realidad Virtual, como por ejemplo técnicas de interacción o configuraciones para ambientes de inmersión. Estas investigaciones dan comienzo al proyecto “Low Cost Cave”, buscando fortalecer el laboratorio para facilitar este tipo de desarrollos.

Estas aplicaciones son un poco complejas en su desarrollo, debido a que el tiempo requerido para la instalación y configuración de las herramientas es alto, tiempo que se incrementa con el aprendizaje de las mismas y que se ve reflejado en desarrollos lentos y poco estables. Para intentar resolver en una primera medida estos problemas se puede utilizar diferentes ambientes de trabajo, evitando todo el proceso de configuración e instalación, pero estos ambientes son complicados de usar, requieren de un conocimiento

[3] CAD: Diseño Asistido por Computador

técnico importante; están dirigidos a cierto tipo de configuración computacional; y en algunos casos tienen un costo muy elevado.

Es por esto que nace la idea de crear una infraestructura que permita el desarrollo de aplicaciones de Realidad Virtual a un costo menor, utilizando los dispositivos con los que cuenta la universidad, las instalaciones y el software disponible de libre acceso (GNU), para crear un ambiente de trabajo lo suficientemente completo y flexible para la creación de este tipo de aplicaciones.

# 2. ESTADO DEL ARTE DE LAS HERRAMIENTAS DE DESARROLLO DE SOFTWARE DE RV

Durante la evolución de la computación grafica y la realidad virtual se han desarrollado diferentes tecnologías tanto de software como de hardware que permiten crear con mayor facilidad este tipo de software. Como consecuencia de este proceso, actualmente existe una gran cantidad de herramientas útiles para el desarrollo de una aplicación de RV.

Cada una de las herramientas tiene sus beneficios, hay unas herramientas mucho mas completas que otras, pero en general se busca que el desarrollo de una aplicación no sea tan traumático; se busca encapsular cada uno de los componentes que involucra la RV y ofrecerlos como un elemento mas del API (application programming interface), de desarrollo. A continuación, se realizará un breve resumen sobre las herramientas de software, plataformas de desarrollo, metodologías y procedimientos, la cual servirá para establecer los elementos apropiados para la propuesta desarrollada en está investigación.

## 2.1. Herramientas de Software

Como se comentó anteriormente, existe una gran diversidad de herramientas de software que nos ofrecen diferentes opciones a la hora de desarrollar una aplicación de RV, por lo cual se intenta mostrar las más representativas y una breve descripción de su utilidad.

### 2.1.1. SYSYGY

La herramienta SYSYGY [SCH 03, SYS 03], fue desarrollada por el ISL (Integrated

Systems Laboratory) de la Universidad de Illinois. Su principal objetivo es soportar la mayor cantidad de aplicaciones de RV en un sistema distribuido, obteniendo con esto un mayor rendimiento que el obtenido por una máquina SGI Onyx[4]. La idea es montar un sistema distribuido heterogéneo que pueda utilizar cualquier configuración de máquinas y cada una de ellas pueda cumplir con alguna labor. Por ejemplo, máquinas como las x86 o las MIPs puedan ser útiles nuevamente.

Este sistema está basado en una capa de mensajes con los cuales se puede establecer la comunicación entre diferentes elementos distribuidos que se están procesando, tales como gráficos, sonido, administración y acceso a los datos, y dispositivo I/O. SYSYGY permite centralizar la configuración y administración de los diferentes elementos, siendo ésta su fortaleza.

Como se ve el la siguiente imagen (Imagen 1), SYSYGY divide toda su estructura en cuatro capas:

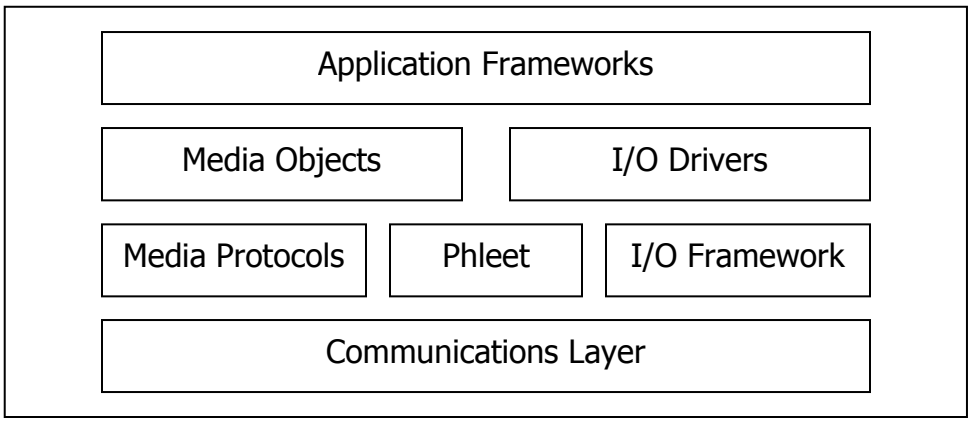


**Imagen 1 Plataforma de SYSYGY [SYS 03]**

[4] Servidor Ofrecido por Silicon Graphics.

- Primera Capa: Conocida como la "communications layer", es la capa más baja y está relacionada con los protocolos de comunicación y con la manipulación de la conexión, desconexión y reconexión de las máquinas en el sistema distribuido.

- Segunda Capa: Está dividida en tres partes, la "media protocols" encargada de la creación de protocolos, incluyendo opciones de sincronización y comunicación entre dispositivos, el "Phleet" que simula un sistema distribuido y ofrece opciones de administración para los procesos que están registrados en el, y por ultimo "I/O framework" encargado de la administración de los dispositivos de entrada y salida.

- Tercera Capa: La componen los "media objects" y los "I/O Drivers", que permiten la manipulación de los dispositivos, por ejemplo, trackers, guantes, cascos, etc…

- Cuarta Capa: Está última capa conocida como la "Application Frameworks" es el la interfaz del ambiente de desarrollo.

### 2.1.2. DICELIB: Distributed Cave Engine Library

Creado en la Universidad de Sao Paulo por Bruno Barberi Gnecco, Dicelib [BAR 03], es una librería para trabajar sistemas distribuidos que requieren una taza de actualización alta, como aplicaciones gráficas o sistemas CAVE[5].

Tiene como ventajas que está completamente implementada en C++ sobre sockets, lo que permite soportar los protocolos de comunicación más comunes (TCP, UDP, XTP). Ésta implementación es demasiado técnica y requiere de un conocimiento relativamente amplio del lenguaje C.

[5] CAVE: sigla establecida para referirse a un sistema de Realidad Virtual basado en proyecciones, que logra una inmersión completa del usuario al mundo

### 2.1.3. VrJuggler

Vrjuggler [VRJ 03], es un API creado en la Universidad de Iowa, para el desarrollo de aplicaciones gráficas. Su objetivo es abstraer todos los componentes que están involucrados en una aplicación gráfica, tales como sonido, trackers, dispositivos de entrada y salida, etc.

Este API es hecho completamente en código libre y permite la configuración de cada uno de los dispositivos por medio de archivos XML, los cuales son posibles de editar en cualquier editor de texto o con una aplicación en java suministrada con el software, la cual a diferencia de los demás elementos de este API, no es muy estable porque se encuentra en un desarrollo muy temprano.

En la actualidad la comunidad de programadores en VrJuggler es la más activa frente a los demás grupos de desarrollo sobre aplicaciones de RV, principalmente por medio de los foros de discusión. La última versión estable de Vrjuggler es la 1.1 y en desarrollo se encuentra la 2.0 alpha 4, dentro de la cual se tiene la posibilidad de crear un ambiente distribuido, llamado Cluster Juggler. La arquitectura de VrJuggler se muestra en la imagen (Imagen 2) a continuación.

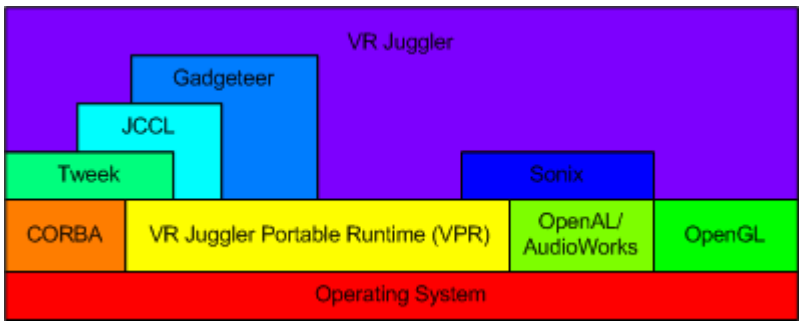


**Imagen 2 Arquitectura de VrJuggler [VRJ 03]**

Simultáneamente con el desarrollo de VrJuggler hay otros desarrollos que apoyan el sistema con el modelamiento de diferentes aspectos, y son los subproyectos patrocinados por el mismo grupo, dentro de estos se encuentran los que se mencionan a continuación.

- JCCL (Librería de configuración para VrJuggler).
- Gadgeeter (Soporte a dispositivos I/O).
- VPR (VR Juggler Portable Runtime Layer).
- Sonix (Objetos de Sonido).
- Tweek (Aplicaciones Graficas Portables).

### 2.1.4. Performer

Performer [SGI 04], es un potente API creado por SGI basado en la manipulación y visualización de grafos de escena. Performer está diseñado para aplicaciones de alto rendimiento, como visualización en tiempo real o animaciones 3D. Permite sincronización entre diferentes máquinas a un alto rendimiento. En la actualidad existen implementaciones para diferentes sistemas operativos y su ejecución tiene como base la comunicación por medio de pipes.

Es un software comercial, pero permite el trabajo con una versión de evaluación la cual cuenta con todas las capacidades de la versión completa.

### 2.1.5. CAVELIB

CAVELIB [VRC 04], es la primera librería creada por Carolina Cruz Neira y Dave Pape en el EVL (Electronic Visualization Laboratory), es mostrada al público en la presentación del primer ambiente tipo CAVE en el congreso SIGGRAPH92. Ayudó a sus desarrolladores a

utilizar de manera más fácil los dispositivos de entrada, los sistemas de visualización y a independizar el procesamiento de los dispositivos.

En la actualidad esta librería es propiedad de VRCO[6], por esta razón es necesario adquirir la versión comercial para obtener todo su potencial.

### 2.1.6. VRPN: Virtual-Reality Peripheral Network

VRPN [VRP 04], es un "framework" desarrollado en C++, que busca independizar los dispositivos de RV de la aplicación. Cuenta con una gran variedad de implementaciones para diferentes dispositivos, los cuales se pueden extender por medio de la creación de clases.

Los dispositivos son identificados con un nombre y un índice; la información se puede acceder por medio de una red TCP-UDP/IP.

Está implementada sobre una arquitectura cliente servidor, donde cada cliente es capaz de obtener la información de cada uno de los dispositivos conectándose al puerto establecido por el servidor.

### 2.1.7. Alice

Desarrollado por la Universidad de Carnegie & Mellon y la universidad de Virginia, Alice [ALI 04], es un ambiente de programación con una interfaz amigable para el usuario, está pensado para aplicaciones sencillas y una simple interacción como un clic en el Mouse o utilizar algún carácter del teclado.

[6] VRCO: Compañía que surge en 1996 y desde ese momento se convierte en la pionera en comercialización de hardware y software que permita I3D (interactive 3D)

Permite crear aplicaciones de RV con mucha facilidad sin necesidad de tener un amplio conocimiento del objeto 3D, dado que este proporciona un conjunto de objetos predefinidos que pueden ser agregados a la escena que se está creando. Una vez se tienen una escena completa se pueden agregar pequeños scripts asociados a estos objetos, para generar animaciones o comportamientos sencillos.

### 2.1.8. WTK: World Toolkit

Es una librería desarrollada en C/C++, para el desarrollo de aplicaciones portables, soporta una gran variedad de hardware y diferentes formatos para la descripción de la geometría de los objetos a representar.

La portabilidad de las aplicaciones, es soportada por una optimización para cada una de las plataformas dentro de las cuales se encuentran SGI, Solaris, HP, DEC, Intel, y Linux. La última versión soporta un ambiente distribuido, así como corrección a varios errores de sus versiones anteriores.

### 2.1.10 Comparación de Herramientas

Cada una de las herramientas anteriormente descritas tiene una fortaleza en alguno de los aspectos que involucra la computación gráfica y los ambientes de realidad virtual. Este capítulo se complementará con un pequeño cuadro que analiza las herramientas en diferentes aspectos y su importancia en cada uno de estos. La tabla describe el lenguaje de programación sobre la cual fue desarrollada la herramienta y su importancia en cada una de las áreas relevantes (representado por un número en donde el 3 es el más significante y el 1 menos significante, adicionalmente habrán casillas con el valor 0 para las herramientas que no de facilidades en esa área).

**Tabla 1 Comparación de las Herramientas**

| HERRAMIENTA | Lenguaje | FU | SD | IC | AD | DG | C |
|---|---|---|---|---|---|---|---|
| **CAVELIB** | C | 1 | 3 | 3 | 3 | 0 | 1 |
| **PERFORMER** | C++ | 1 | 1 | 3 | 2 | 3 | 2 |
| **ALICE** | Python | 3 | 1 | 0 | 0 | 1 | 3 |
| **VRJUGGLER** | C++ | 1 | 3 | 3 | 3 | 0 | 3 |
| **SYSYGY** | C++ | 1 | 2 | 2 | 3 | 0 | 3 |
| **DICELIB** | C++ | 1 | 1 | 3 | 2 | 0 | 3 |
| **VRPN** | C++ | 1 | 3 | 3 | 0 | 0 | 3 |
| **WTK** | C++ | 1 | 3 | 3 | 2 | 3 | 3 |

Los campos de la primera fila en la tabla son abreviaciones que representan las siguientes áreas de interés.

- (FU) Facilidad de uso
- (SD) Soporte de Dispositivos
- (IC) Instalación y Configuración
- (AD) Ambiente Distribuido
- (DG) Detalles en la Geometría
- (C) Costo

Como se puede observar una de las herramientas más equilibradas es VrJuggler, por lo cual se decide que está sea la base de integración de la plataforma.

Existen otras herramientas de software para el apoyo de aplicaciones de RV, pero esta investigación se basa en las nueve anteriormente mencionadas, por considerarse como las mas relevantes y utilizadas.

## 2.2. Plataformas

Así como existen en la actualidad una diversidad de software especializado para la generación de aplicaciones de RV, existen a su vez diferentes opciones en el mercado de plataformas completas para este fin. Sin embargo, estas opciones son comerciales y la información que se encuentra es incompleta y altamente técnica.

A continuación se muestran un subconjunto representativo de ellas y una descripción de sus cualidades.

### 2.2.1. ARSBOX

El ARSBOX [ARS 03], es una solución comercial desarrollada por Ars Electronica, definida como una solución CAVE, está basada en computadores personales (PC) sin un hardware especializado solo usando tarjetas de video comerciales como son las de NVDIA. Este sistema permite visualizar hasta 64 proyecciones simultaneas, se puede adaptar para un sistema CAVE donde se obtiene una visualización estéreo lograda utilizando 2 equipos por proyección.

La interacción con el usuario es lograda a través de una IPAQ (Agenda digital Compaq) con la cual es posible administrar todas y cada una de las visualizaciones del ARSBOX.

### 2.2.2. Sara Virtual Reality facilities

Sara [SAR 03], es una empresa Holandesa dedicada a proveer diferentes plataformas tecnológicas en diferentes ámbitos. Dentro de los productos que ofrece la compañía se encuentra la categoría de Virtual Reality Facilities, dentro de la cual ofrece diferentes tipos de infraestructuras para el desarrollo de aplicaciones de realidad virtual.

Existen diferentes opciones como equipos SGI Onyx, inmersa desk2 y en particular ofrece el servicio de CAVE, el cual incluye toda la infraestructura de software y de hardware necesaria.

El CAVE está dividido en tres subsistemas, procesamiento, visualización e interacción.

- Procesamiento: Logrado por medio de un equipo SGI Onyx 2 Reality Monster.
- Visualización: La proyección está basada en espejos de reflexión y proyectores (Electrohome Marquee 8500) capaces de dar resolución 1280 x 1024 a 120 Hz, lo que permite la visualización estéreo.
- Interacción: De acuerdo a la necesidades del usuario se ofrecen diferentes posibilidades como trackers de movimiento (flock of birds), guantes (Cybergloves) o un joystick (wand mouse 3d).

### 2.2.3. VISBOX

Esta solución [VIS 03] busca la inmersión por medio de una pantalla con visualización estéreo. La plataforma está compuesta por una pantalla de 10 pies de diagonal un servidor SGI Onix 2 requerido para el procesamiento y viene equipado con un paquete de software para desarrollo, ejemplos y documentación.

### 2.2.4. SGI Reality Center

Dentro de los servicios que ofrece SGI se encuentra un grupo que está encargado de la generación de espacios de RV, entre los que se encuentran ambientes tipo CAVE, Fish Tank Desktop, etc…

### 2.2.5. FAKESPACE

FAKESPACE Systems [FAK 04] es una empresa dedicada a proveer diferentes soluciones involucradas con la computación gráfica y la realidad virtual. Dentro de sus productos tienen las soluciones más completas para estos ambientes como lo son las tipo CAVE y las POWERWALL[7].

A diferencia de otras empresas, está cuenta con su propio software para trabajar con su plataforma, FakeSpace Interaction Engine es el motor que permite integrar todos los dispositivos que suministran y centralizar todas las entradas en un servidor, el cual se encarga de enviar la información a todas los elementos del sistema que la necesiten.

### 2.2.6. BARCO

BARCO [BAR 04], es una empresa dedicada a ofrecer soluciones relacionadas con visualización a diferentes niveles. Las mas relacionadas con el tema de realidad Virtual y computación grafica son “Large screen visualization” y “Visualization for life-critical decision making”, destacándose en esta última las soluciones que ofrecen para el despliegue de imágenes médicas.

### 2.2.7. MULTIGEN

Formada en septiembre de 1998, MultiGen Paradigm [MUL 04] surge como resultado de la unión entre de dos compañías, MultiGen INC y Paradigm Simulation INC. Esta compañía está enfocada en los desarrollos que involucran simulación o que necesiten procesamiento gráfico en tiempo real.

[7] Esta configuración consiste en una pantalla de cristal líquido de grandes dimensiones, en la cual se puede interactuar con dispositivos no convencionales

Dentro de sus soluciones ofrecen diferentes configuraciones de hardware y software y lo dividen en cuatro categorías: Database, Runtime, 3DGis y Standard, dentro de las cuales se clasifican los diferentes requerimientos de cada uno de los ambientes.

En la actualidad existen empresas dedicadas a temas puntuales de la Realidad Virtual, como visualización, dispositivos, infraestructuras, etc. Las plataformas anteriormente descritas se consideraron como las más completas, por lo cual permiten dar una aproximación a la configuración de la infraestructura a la que se desea llegar.

## 2.3. Metodologías de desarrollo para Realidad Virtual

En la actualidad no existen muchas metodologías para el desarrollo de software de RV, Por lo cual se hará una breve descripción de las más importantes.

### 2.3.1. MARYGOLD

MARYGOLD [WIL 04], fue desarrollada por James William y Michael Harrison en la Universidad de York con el objetivo de entender, modelar, y crear los prototipos de la dinámica en los ambientes virtuales. Esta metodología se basa en la división de los ambientes virtuales en dos tipos de componentes dinámicos: las técnicas de interacción y los objetos del mundo, los primeros relacionados con los dispositivos de entrada manipulados por el usuario y los segundos los objetos que el usuario percibe y puede manipular.

### 2.3.2. PMIW

En 1999 Robert Jacob, encabeza la propuesta de PMIW [JAC 99], la cual describe una aplicación como una combinación de dos elementos: un flujo de datos que describe el flujo

de información desde los dispositivos de entrada y una máquina de estados que describe los diferentes modelos de ejecución.

El comportamiento de las aplicaciones está dado por eventos desde la interfase y la ejecución de cada nodo en el flujo de datos depende de la máquina de estados.

Esta notación proporciona bastante detalle para representar una técnica de interacción, pero requiere de muchos conceptos técnicos. Los modelos probados en ese momento eran relativamente sencillos y faltaría validar este modelo en aplicaciones mucho más complejas que las utilizadas.

## 2.4. Arquitecturas de desarrollo para Realidad Virtual

Las metodologías de desarrollo promueven la aparición de diferentes arquitecturas las cuales permiten un desarrollo de software estructurado y esto logrado a través de una serie de patrones que se establecen en la arquitectura, a continuación se mostraran diferentes arquitecturas que se pueden adaptar para el desarrollo de aplicaciones de RV

### 2.4.1. MVC & PAC

Estas dos arquitecturas no son propias para el desarrollo de software de realidad virtual sino para cualquier tipo de software, pero contienen los elementos necesarios para adaptarse al desarrollo del tipo de aplicaciones que se desean. Ambas metodologías están enfocadas en la interfaz del usuario, y las aplicaciones de realidad virtual son un caso en particular de esto.

La primera arquitectura, Modelo Vista Controlador (MVC) [VIC 04], como su nombre lo indica está dividida básicamente en tres elementos que representan el sistema:

- Modelo: Este primer elemento es el encargado de generar el modelo, todos los elementos que componen el mundo 3D, por lo cual debe permitir formas de manipularlo.
- Controlador: Este elemento se encarga de recopilar todos los cambios que los diferentes actores desean hacerle al mundo y realizarlos ordenadamente, adicionalmente se debe encargar de mantener la sincronización entre el mundo y todas sus visualizaciones.

- Vista: Este último elemento es el encargado de mostrar el estado del mundo por medio de alguna visualización, permitiendo así que existan diferentes vistas del mismo modelo

Como se observa se tiene una separación entre el comportamiento del mundo y las visualizaciones, es por esto que esté modelo se puede aplicar en el desarrollo de este tipo de aplicaciones, separando el procesamiento que conllevan los cambios en el modelo, de la visualización final que obtiene el usuario.

La otra arquitectura "Presentation Abstraction Control" (PAC) [VIC 04], se basa en la definición de agentes que se responsabilizan en el desarrollo de alguna tarea específica en la aplicación, clasificándolos en 3 tipos:

- Abstracción: Son los agentes que se encargan de obtener la información, en el caso de las aplicaciones de realidad virtual, se obtienen los datos de los diferentes dispositivos de entrada.
- Control: A diferencia de los anteriores agentes, estos buscan modificar la información que reciben por medio de mensajes y enviarla modificada nuevamente.
- Presentación: Estos Agentes son los encargados de la visualización, si se compara contra el modelo MVC serian los encargados de las vistas.

Las características propias de las metodologías mencionadas se prestan para que la visualización este separada del procesamiento de la información, elemento primordial en la optimización de las aplicaciones de RV.

### 2.4.2. MARYGOLD Toolset

Para poder trabajar con la metodología establecida en MARIGOLD [WIL 04], se ofrece una herramienta que la soporta, ésta esta dividida en tres utilidades las cuales se mostraran a continuación:

- Hybrid specification builder (HSB): Esta utilidad permite desarrollar diagramas de flujo que especifican la dinámica del sistema, uniendo los componentes que se definieron (dentro de estos se encuentran las técnicas de interacción).
- Complex object builder (COB): Ya que se tiene la dinámica del sistema, es posible generar los modelos geométricos que van a ser visualizados por medio de esta utilidad. Dentro de las opciones que permite, se encuentra la utilidad de obtener modelos de terceros, en particular obtener modelos de 3D Studio.
- Prototype builder (PB): Esta utilidad es la encargada de integrar la visualización generada en COB con las técnicas de interacción generadas por en HSB

### 2.4.3. InTml: Interaction Techniques Markup Language

InTml [FIG 04], es un lenguaje para la descripción de diferentes aplicaciones de realidad virtual como presentaciones de escritorio 3D, realidad aumentada y en general cualquier aplicación que involucre dispositivos de entrada, de salida y técnicas de interacción. InTml permite describir de una manera sencilla las clases para interacción 3D y la administración de dispositivos. A su vez permite que los desarrolladores combinen estos elementos para lograr la creación de una aplicación.

Esta descripción es posible dado que las aplicaciones de RV pueden ser vistas como un conjunto de filtros que por medio de sus entradas y salidas pueden estar interconectados. Una descripción mas completa de esta arquitectura se hará el apéndice Arquitectura InTml.

Como se puede apreciar InTml es una metodología lo suficientemente completa y estructurada, porque establece una clara división entre el desarrollo de una aplicación y la infraestructura que la soporta, estableciendo parámetros bien definidos en cada una de ellas. Adicionalmente, al dividir el desarrollo de las aplicaciones RV en capas esta metodología se convierte en una herramienta útil para alcanzar uno de los objetivos de esta investigación, como es el facilitar el trabajo de los diseñadores de aplicaciones de RV, ya que no necesitan tener un conocimiento técnico del hardware y software utilizado.

## 2.5. Requerimientos

En la actualidad, no existe ningún estándar de requerimientos para seguir en el desarrollo, uso y mantenimiento de aplicaciones de realidad virtual. Dentro de los desarrollos del laboratorio de computación gráfica y realidad virtual de la Universidad de los Andes se han logrado definir una serie de elementos que deben tenerse en cuenta durante el desarrollo y uso de las aplicaciones por lo tanto se desean tener en cuenta para el ambiente de desarrollo propuesto.

Los requerimientos que se han logrado identificar en desarrollos anteriores del laboratorio se mostraran a continuación

- Deben existir mecanismos de calibración para los diferentes dispositivos con los que se cuenta.
- Es necesario que el conjunto de dispositivos de entrada y salida de una aplicación sean fácilmente configurables.
- Debe ser posible utilizar dispositivos remotos, así como simular un dispositivo mediante otros, o cambiar totalmente la interfaz hardware de una aplicación.

- Se debe permitir el cambio de técnicas de interacción, como un mecanismo para probar diversas alternativas en el diálogo con el usuario y como forma de adaptar de manera óptima nuevos dispositivos de interacción.
- Debe poderse elegir la calidad de las gráficas mostradas dependiendo de las capacidades computacionales de la máquina donde corra la aplicación.
- Se debe poder contar con una librería de acceso a diversos dispositivos de entrada/salida, así como diversos tipos de contenido y diversas técnicas de interacción.

La lista anterior contempla los elementos mínimos que deben ser establecidos en los procedimientos de desarrollo, uso y mantenimiento de aplicaciones RV, que han surgido con la experiencia del laboratorio de informática gráfica y realidad virtual de la Universidad de los Andes, sin embargo estos elementos puede complementarse en la medida en que se valla madurando la infraestructura o con el intercambio de experiencias con otros laboratorios.

Hasta el momento, se han mostrado los diferentes elementos que en la actualidad ayudan a los desarrollos de aplicaciones de realidad virtual y su estado. En la siguiente sección, se dará inicio a la especificación de todos los elementos que se han decidido utilizar para conformar la infraestructura que esta investigación propone y una descripción detallada de su funcionamiento en la infraestructura.

# 3. DEFINICIÓN DE LA PROPUESTA DEL AMBIENTE DE DESARROLLO

Como se mostró en la sección anterior en la actualidad existe una gran variedad de plataformas y software para el desarrollo de aplicaciones de RV, que permiten crear diferentes configuraciones. Esto es posible gracias al intento de las herramientas por abstraer los dispositivos y aunque este intento es muy útil aun persisten ciertos problemas, tanto en las plataformas como en el software, que se mencionaran más adelante y que buscan ser solucionados con la propuesta de la infraestructura.

El problema principal de las plataformas existentes son los costos, dado que la mayoría están basadas en equipos de procesamiento gráfico con grandes capacidades, es el caso de las configuraciones como el Sara Cave o VISBOX mostradas anteriormente. Estos costos pueden ser superiores a los US$500.000, por lo cual en la realidad colombiana no son viables. El sistema que se plantea debe buscar alternativas mucho más económicas que las plataformas actuales en el mercado, utilizando equipos con menores características, pero que permitan obtener buenos resultados.

Por otra parte, el software para el desarrollo de aplicaciones de realidad virtual tiene como problema que para la generación de una aplicación se necesita el conocimiento de un API o de un lenguaje de programación específico. Es por esto, que se desea crear una separación entre la tecnología usada y el desarrollo de una aplicación, lo cual se logro por medio de InTml.

Basados en la identificación de los problemas anteriores, este capitulo muestra la descripción del hardware disponible en el laboratorio que hace parte de la infraestructura y la configuración de software que soporta este hardware y que intenta solucionar estas

dificultades. Por último se especificarán todos los procedimientos necesarios para el uso de la plataforma.

## 3.1. Hardware

La infraestructura planteada debe permitir configurar diferentes ambientes en los que las aplicaciones de RV pueden llegar a utilizarse, en particular los ambientes con los que cuenta el laboratorio de informática gráfica de la universidad de los Andes. A continuación se mostrará la configuración de los dispositivos independizándolos del sistema y se hará una breve descripción de los ambientes con los que cuenta la Universidad mostrando algunas de sus características, el grafico a continuación muestra la estructura actual del laboratorio (Imagen 3).

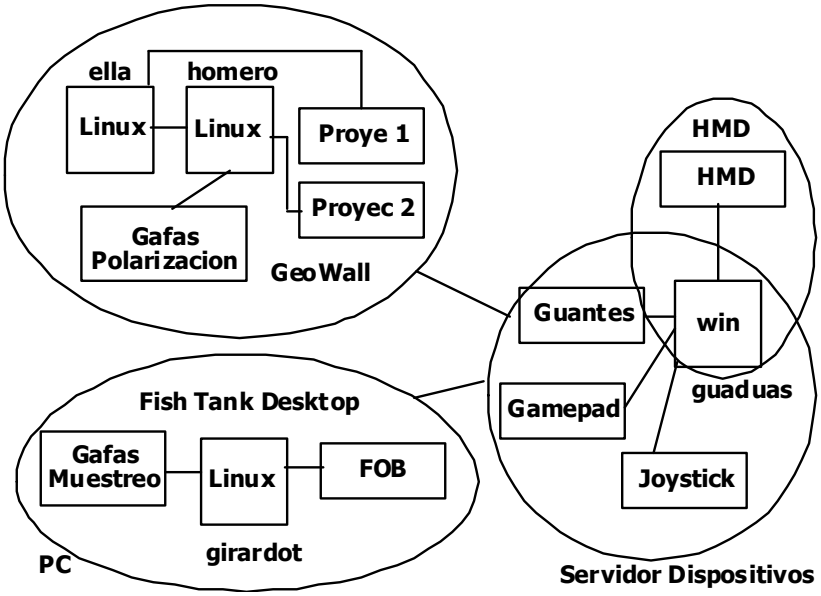


**Imagen 3 Configuración del Laboratorio**

### 3.1.1. Servidor de dispositivos

El sistema que se crea debe ser extensible y uno de los principales elementos para permitir esto es la posibilidad de agregar nuevos dispositivos de entrada en el banco de trabajo. Para esto se decidió independizar el sistema de procesamiento de los dispositivos, montando un servidor de dispositivos.

Cuando se configura el servidor de dispositivos es posible acceder a la información generada por medio de una red convencional TCP/IP. En la actualidad el software utilizado como servidor de dispositivos es VRPN, el cual fue descrito en la sección 2.1.6, y se tienen registrados los siguientes elementos:

- trackers
- guantes
- joystick's
- gamepad's.

Se desea poder configurar todos los dispositivos del laboratorio en un conjunto de máquinas destinadas específicamente para esta labor, permitiendo distribuir las cargas entre los diferentes dispositivos.

### 3.1.2. HMD

Dentro de los ambientes mas utilizados en aplicaciones de RV, se encuentran los basados en dispositivos HMD (Head Mounted Display), debido a que logran una buena inmersión de los usuarios en el mundo simulado. Este ambiente es complementado con elementos de interacción, dentro de los cuales se encuentran principalmente los joystick, los guantes y

los gamepad, permitiendo al usuario comunicarse con el mundo, ya sea navegándolo o modificando.

Los HMD están basados en el concepto de parallax, este nombre se le da a la diferencia entre los ángulos de visualización generados por dos puntos de vista ligeramente diferentes. La idea de estos dispositivos es generar dos versiones diferentes del mundo y ubicarlas en frente de cada ojo, manteniendo estas visualizaciones sincronizadas y logrando con esto engañar a la mente, ya que está genera una visión de profundidad en el modelo, en la actualidad el laboratorio cuenta con unas virtual I/O glasses, pero solo se genera una imagen, por lo cual este efecto no se utiliza.

Aunque este ambiente es muy completo tiene una serie de problemas, el más importante es generado por el dispositivo como tal, debido a que el usuario debe soportar el peso del casco, el usuario se fatiga mas rápidamente por la carga adicional y en algunas ocasiones puede llegar a sentir mareo. Adicionalmente a estos problemas, se tiene que el sistema está conectado por medio de cables al computador, por lo que el desplazamiento es limitado por la longitud de los cables, sin tener en cuenta que el usuario puede pisar o enredarse con los cables al momento de caminar [PER 04].

### 3.1.3. Fish Tank Desktop

A diferencia del anterior, este ambiente es basado en una pantalla CRT (Tubo de Rayos Catódicos) con muy buenas características, como su nombre lo indica se busca convertir el monitor en una pecera, es decir utilizar el monitor como una ventana al mundo virtual 3D. Esta visualización se logra por medio de una buena tarjeta gráfica pero no se logra una inmersión completa del usuario al mundo virtual. El laboratorio actualmente cuenta con un computador equipado de una tarjeta grafica nvidia gforce 3000 para la visualización y dos procesadores Xeon para el procesamiento.

Al igual que el anterior ambiente éste debe ser complementado por medio de dispositivos de interacción, en la actualidad el dispositivo conectado a este ambiente son tres trackers FOB (Flock of Birds) que permiten interactuar de diferentes maneras con el mundo virtual, uno para la dirección de la cabeza y los otros dos para interactuar.

El problema principal de este ambiente es que no se logra inmersión del usuario en el mundo, primero por que no se logra tener el efecto de proyección 3D y segundo los marcos del monitor rompen la realidad.

La Imagen 4 muestra el ambiente fish tank que utilizamos en el laboratorio.

**Imagen 4 Fish Tank Desktop**

### 3.1.4. Stereo Pc

A diferencia del anterior ambiente este no cuenta con el punto de vista del usuario, pero si se busca utilizar la visión en stereo, lo cual se logra por medio de una gafas de muestreo, estas gafas permiten filtrar las imágenes de acuerdo a un patrón de frecuencia y permiten engañar de una manera mas rustica al ojo humano.

Las características con las que debe contar el monitor que es utilizado en este tipo de ambiente es un refresco mayor a 120 Hz (60 Hz para cada ojo). El laboratorio actualmente cuenta con el sistema "X3D Extreme 3D System", el cual se conecta a la tarjeta de video y sirve de puente entre tarjeta y monitor.

### 3.1.5. PC

Es necesario generar un ambiente que permita hacer diferentes pruebas durante el desarrollo de una aplicación, este ambiente consiste en el uso de un computador convencional, el cual permite desarrollar las pruebas pertinentes a la aplicación sin la necesidad de montar el ambiente completo, en donde finalmente se desea mostrar la aplicación.

### 3.1.6. GEOWALL

Este tipo de ambiente consiste en la configuración de computadores, proyectores y gafas de polarización, el laboratorio se encuentra configurando un cluster de dos maquinas, las cuales proyectaran las imágenes en dos proyectores NEC LT 240K con un filtro de color (azul y rojo), permitiendo que se puede lograr el efecto de volumen, por medio de unas gafas de polarización.

## 3.2. Software

La infraestructura que se ha desarrollado, está soportada sobre un conjunto de herramientas de software descritas anteriormente las cuales permiten integrar cada uno de los elementos disponibles en el laboratorio.

El sistema tiene como núcleo de integración VrJuggler (Imagen 5), dado que éste ofrece interacción con otro tipo de tecnologías como Performer y el acceso a diferentes dispositivos tanto de entrada como de salida, en particular la conexión con un servidor de dispositivos VRPN. Toda la configuración del sistema es dada por medio de archivos XML que contienen la información de cada uno de los dispositivos a usar. Esta facilidad de configuración nos permite generar diferentes ambientes por medio de la creación de archivos que representen cada uno de los dispositivos.

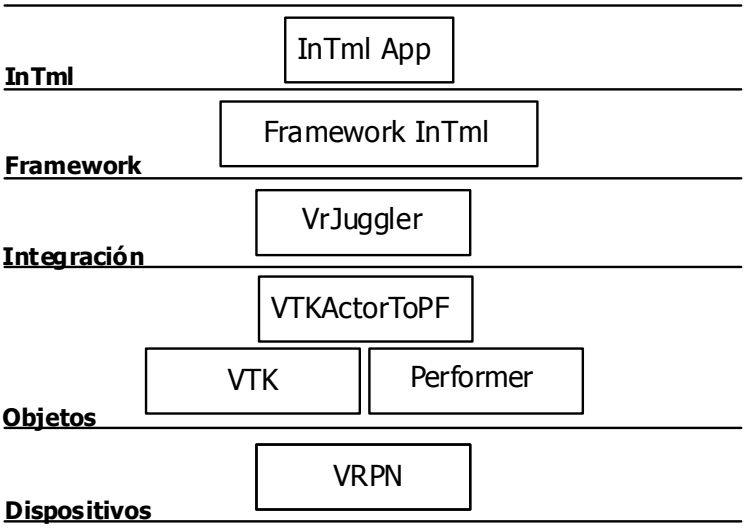


**Imagen 5 Diagrama de Capas de Configuración del Software**

Dado que el desarrollo de una aplicación requiere de un alto conocimiento en lenguaje C++ (sobre el que cual está implementado Vrjuggler) al igual que un conocimiento del API, se busca facilitar el desarrollo de las aplicaciones por medio de un framework InTml adaptado para esta configuración (Vrjuggler + VRPN + Performer), este framework consta de un sistema configurable, que permite acceder a los distintos recursos por medio de archivos de

configuración, los cuales permiten que las aplicaciones cambien entre diferentes configuraciones de ambientes de RV, de acuerdo a las necesidades de la aplicación.

Dada la metodología propuesta por InTml el diseñador que desea utilizar el framework, ya no necesita desarrollar la aplicación completamente sino que la transforma en la unión de diferentes filtros (componentes) los cuales representan dispositivos, técnicas de interacción o contenidos (Modelos 3D). Aunque este es el objetivo final, la implementación actual solo está en la posibilidad de permitir cambios en la aplicaciones con facilidad, ya sea de un ambiente a otro o modificar alguna aplicación para lograr obtener otra, el objetivo final se logrará a medida que se desarrollen más filtros para el sistema, ya que los filtros en la actualidad son limitados y están siendo modificados constantemente para cumplir nuevos requerimientos.

Como se mencionó anteriormente se desea crear un servidor de dispositivos, y aprovechando el soporte con el que cuenta Vrjuggler poderse comunicar con un servidor VRPN, toda la infraestructura está montada sobre una red TCP/IP, en donde los dispositivos se encuentran configurados en una arquitectura cliente – servidor y se pueden acceder desde cualquier máquina del laboratorio. Logrando que éstos sean independientes de la infraestructura de desarrollo. A continuación se muestra un diagrama (Imagen 6) que ilustra lo descrito anteriormente.

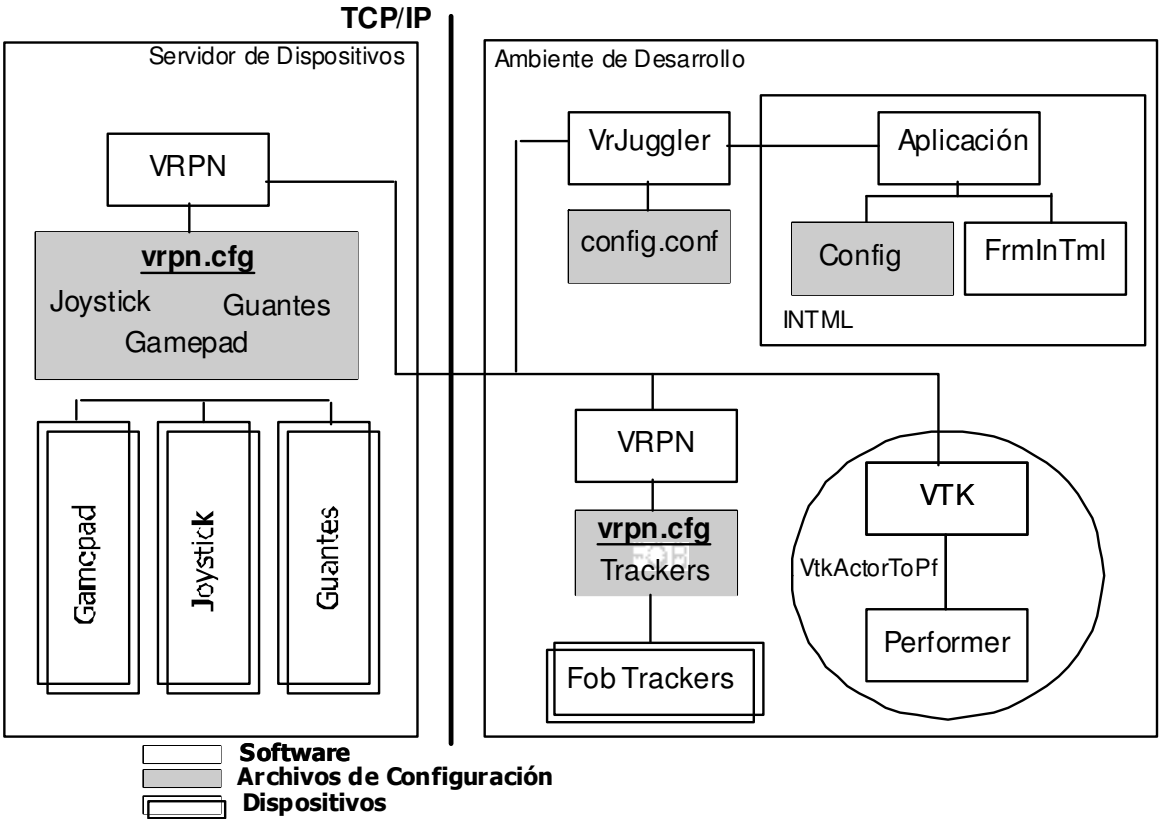


**Imagen 6 Infraestructura de Software**

Para validar la infraestructura se tomó como caso de estudio uno de los desarrollos actuales en el laboratorio, que está relacionado con la reconstrucción de imágenes médicas y en particular las imágenes obtenidas por medio de escáner. Para esta reconstrucción es utilizada una herramienta llamada VTK [VTK 04], la cual permite de una manera relativamente sencilla realizar esta reconstrucción utilizando un conjunto de patrones de gris.

La reconstrucción del modelo en VTK tuvo que ser transformada a un modelo en Perfomer, por lo cual se utilizó una clase implementada por Paul Rajlick conocida como VtkActorToPf [RAJ 04]. La cual transforma el modelo creado por VTK a un nodo en el grafo de escena Performer, siendo así posible integrarlo con VrJuggler de una manera transparente.

Todo este conjunto de herramientas de RV fueron complementadas con un framework InTml, basándose en una versión existente desarrollada por Pablo Figueroa, la cual estaba completamente en C++ y se adaptó para poder soportar la configuración actual del laboratorio VrJuggler + Performer + VRPN. A continuación se mostrará la estructura utilizada en el framework y se especificará el funcionamiento de las clases principales del desarrollo.

El framework se puede dividir en 4 grandes partes, filtros, puertos, información y sistema, los cuales tienen una función específica:

- Filtros: Encargados del comportamiento de los nodos en el sistema.
- Información: Modelan los objetos y los datos que transmitirán los filtros.
- Puertos: Modelan el medio por el cual se comunicarán los filtros.
- Sistema: Encargado de hacer funcionar todo el framework.

A continuación se mostrará a grandes rasgos cada uno de estos.

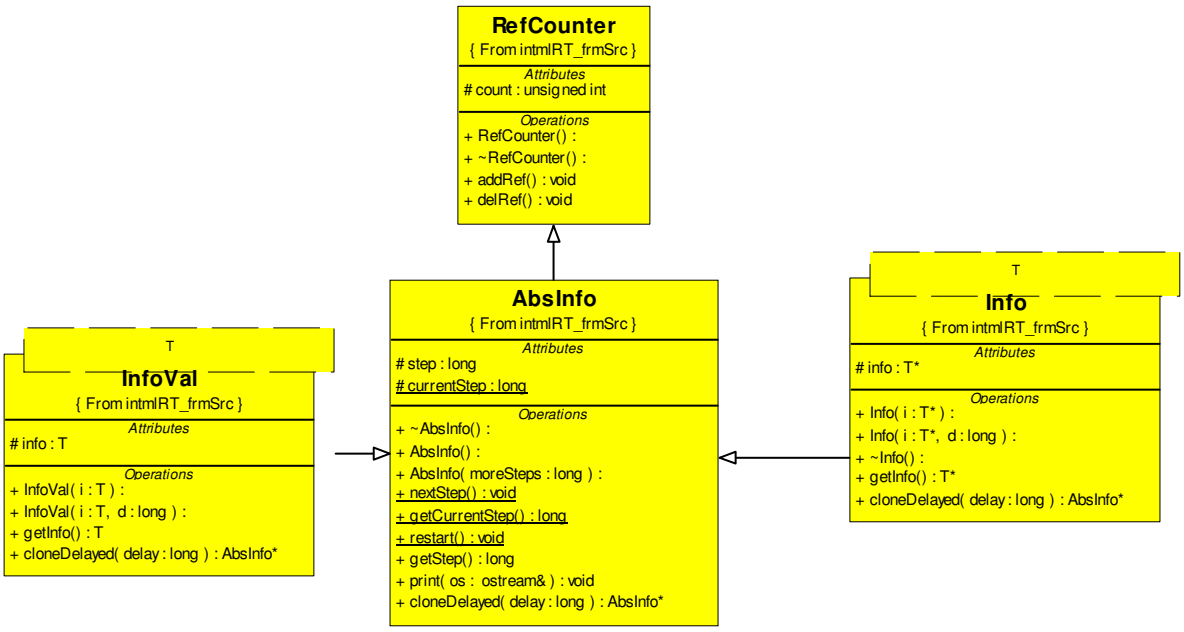


**Imagen 7 Diagrama de Clases Framework InTml (Información)**

**Información:** Como se puede ver en la gráfica anterior el framework define dos templates: Info e InfoVal. InfoVal representa tipos básicos como un entero o un flotante, mientras que Info representa cualquier tipo en el que sea mas eficiente referenciarlo que contenerlo, esta información es la que los filtros transmiten cuando el sistema está en funcionamiento.

Adicionalmente para optimizar el proceso de creación y destrucción de la información se utiliza la clase RefCounter, la cual permite conocer si la información debe ser destruida o no. Para esto, cada vez que un filtro transmite la información incrementa el contador, y cuando la recibe se decrementa, en el momento en que éste llegue a cero significa que la información deja de ser usada. Se podría decir que es el sistema de recolección de basura de todo el framework[8].

[8] En este documento solo hace referencia RefCounter, no se especificara todo el funcionamiento de este.

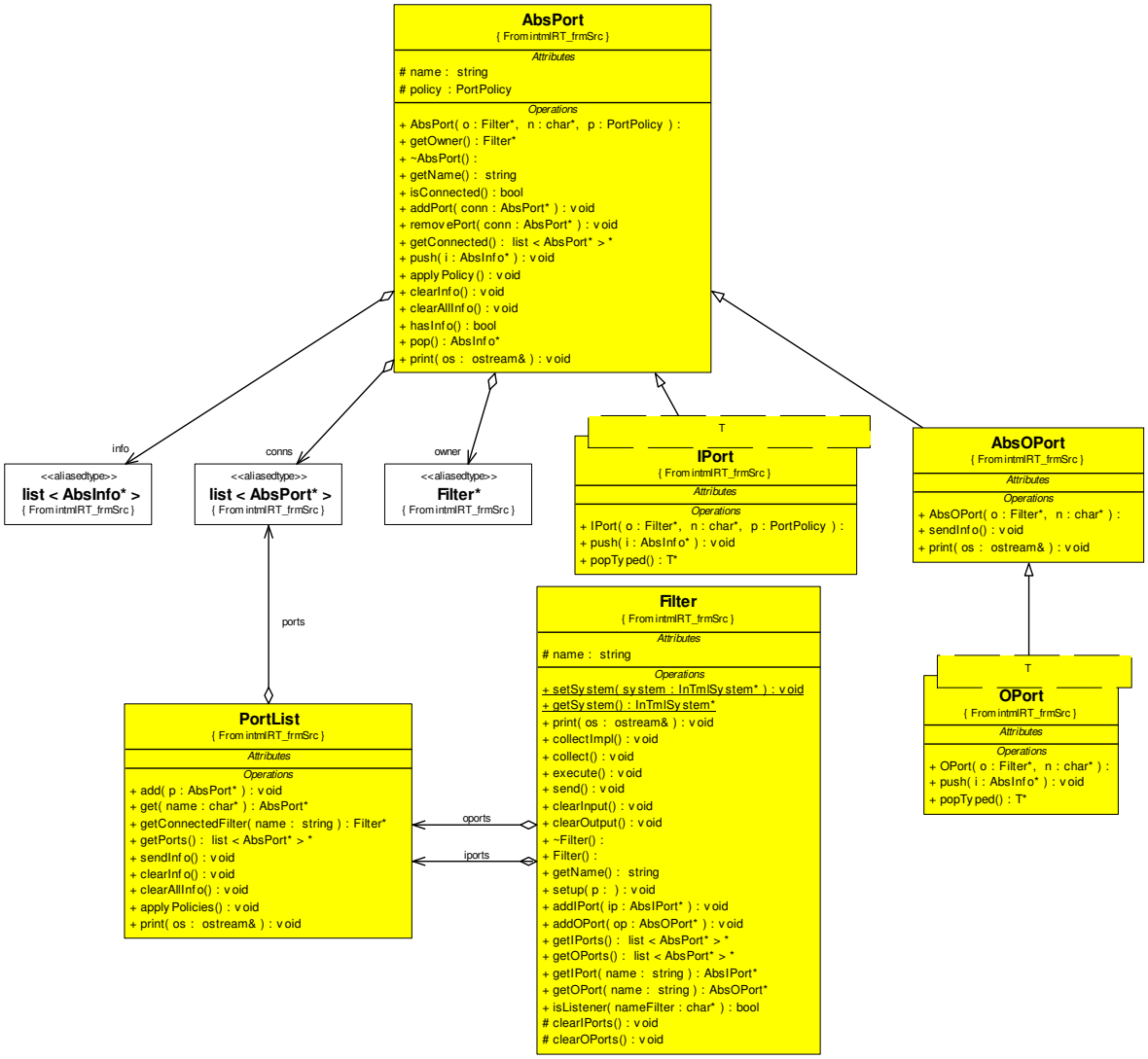


**Imagen 8 Diagrama de Clases Framework InTml (Puertos)**

**Puertos:** Como se mencionó anteriormente los filtros se pueden comunicar por medio de puertos, en la gráfica anterior (Imagen 8) se observa esta relación, la clase Filter tiene dos conexiones con un listado de puertos (iports y oports), los cuales a su vez por medio de la clase AbsPort obtienen la relación con la información..

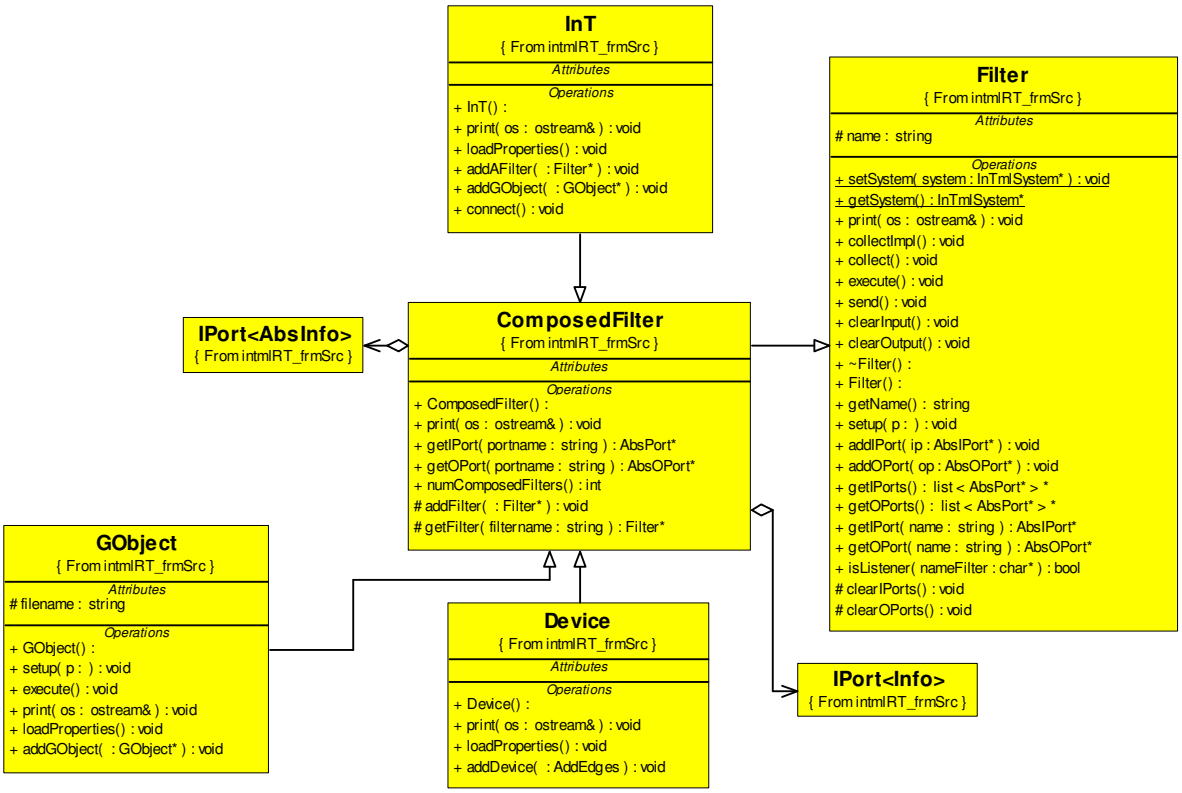


**Imagen 9 Diagrama de Clases Framework InTml (Filtros)**

**Filtros:** Esta es la parte más importante del sistema, dado que todo en el framework InTml es representado por filtros. Se comentó anteriormente que el sistema constaba de tres tipos de filtros, dispositivos, objetos y técnicas de interacción. En el diagrama anterior (Imagen 9) es posible distinguir la diferencia de estos tres en el sistema, por medio de las clases InT, GObject y Device, las cuales a su vez están ligadas directamente con la clase Filter.

Dentro de los métodos de la clase Filter se debe resaltar el método execute, ya que como se verá más adelante este método le da el comportamiento a cada uno de los filtros, y se encarga de mantener la circulación de la información por todo el sistema.

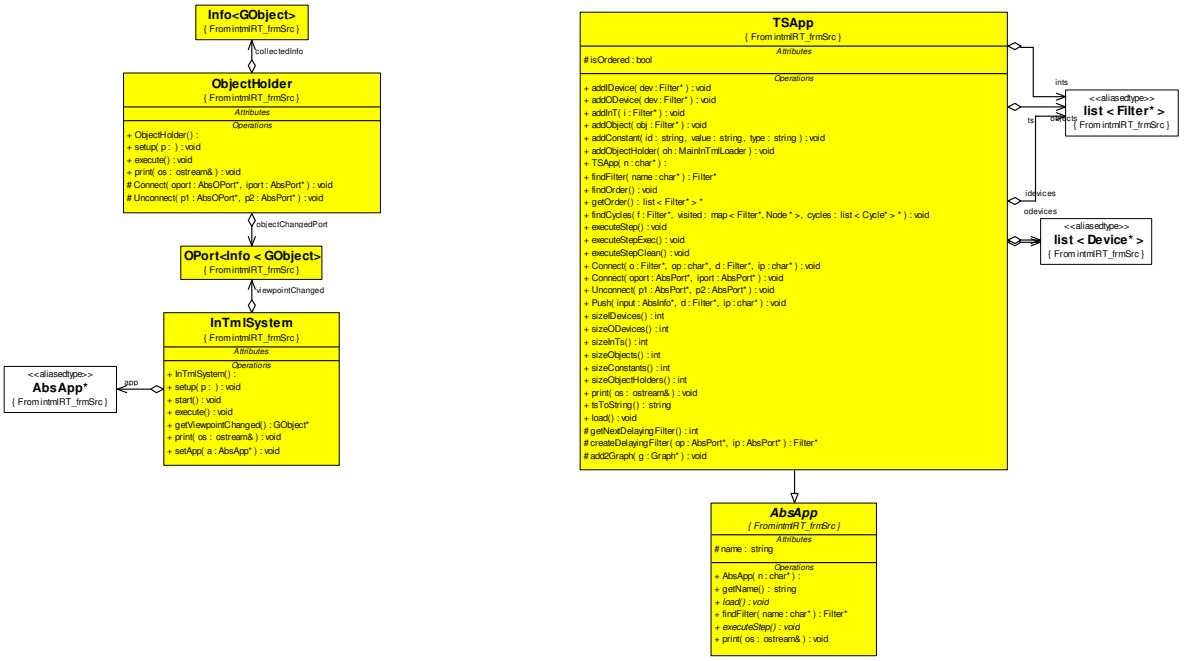


**Imagen 10 Diagrama de Clases Framework InTml (Sistema)**

**SISTEMA**: El sistema consta de dos partes fundamentales, el sistema como tal y la aplicación, representado por medio de las clases InTmlSystem y TsApp (Imagen 10). El primero encargado de establecer todo el sistema y el segundo encargado de ejecutar todo el flujo de la aplicación por medio de la función executeStep();

Para poder utilizar el framework en la infraestructura que se plantea se realizaron las siguientes adaptaciones en el sistema Intml.

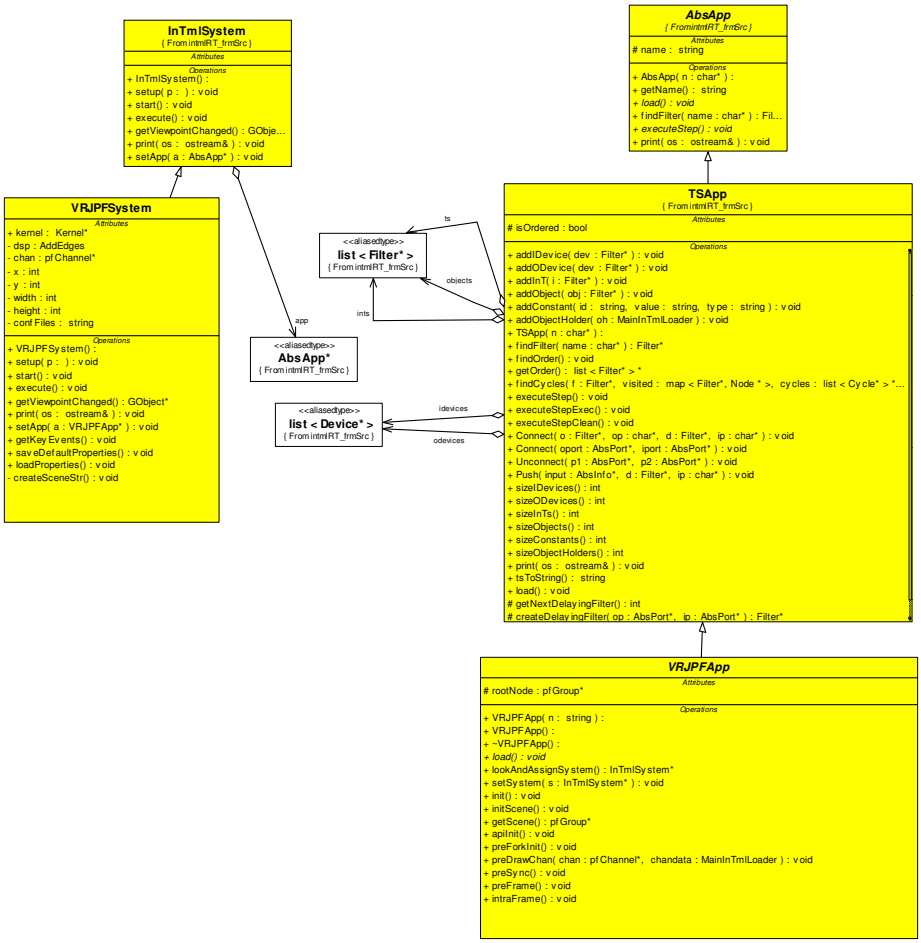


**Imagen 11 Diagrama de Clases Framework Intml (VRJUGGLER + PERFORMER)**

Como se observa en el diagrama anterior (Imagen 11) se crearon generalizaciones para las clases existentes, generando el sistema en particular. Las nuevas clases heredan del sistema en el framework, y permiten crear el ambiente de ejecución MainInTmlLoader este es el encargado de iniciar las aplicaciones (Imagen 12).

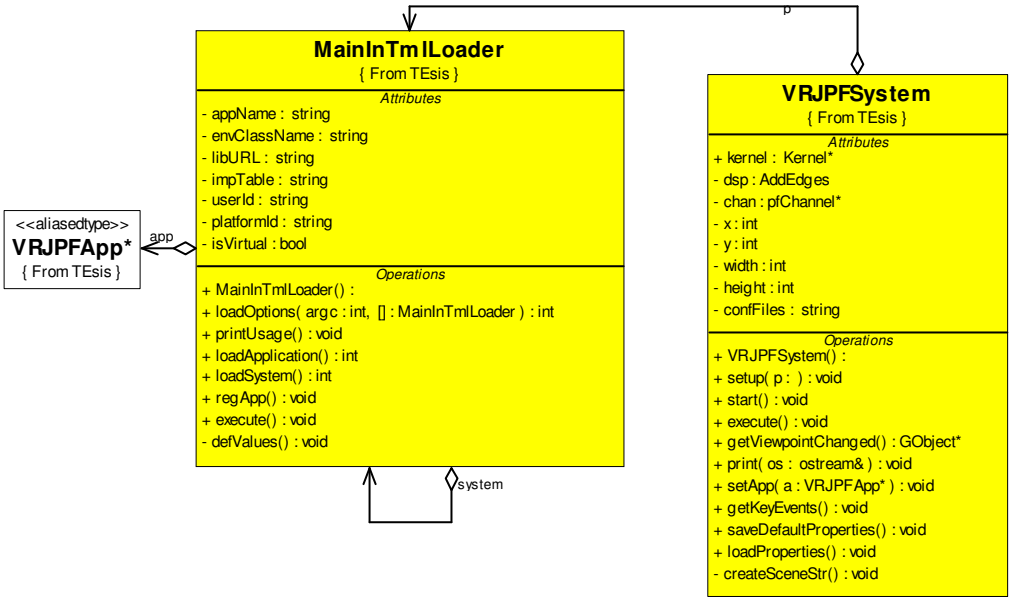


**Imagen 12 Diagrama de Clases Framework Intml (Ambiente de Ejecución)**

El proceso seguido por el sistema para ejecutar una aplicación se muestra en la tabla a continuación y se describe más adelante.

**Tabla 2 Método main de la Clase MainInTmlLoader (MainInTmlLoader.cxx Linea 174)**

```
1  int main(int argc, char* argv()){
2    Properties::setPath("../Properties/");
3    MainInTmlLoader m;
4    int ans = m.loadOptions(argc, argv);
5    if( ans != 0 )
6      return ans;
7    ans = m.loadApplication();
8    if( ans != 0 )
9     return ans;
10   std::cout << "App ready" << std::endl;
11   ans = m.loadSystem();
12   if( ans != 0 )
13     return ans;
14   std::cout << "System ready" << std::endl;
15   m.regApp();
16   std::cout << "Register ready" << std::endl;
17   // run application
18   m.execute();
19 }
```

- Cargar las propiedades: Al momento de comenzar a ejecutarse se cargan las propiedades del sistema, como se puede observar en la línea 2 de la siguiente tabla (Tabla 2).
- Cargar la aplicación: En el método loadApplication se llama a la clase InTmlLoader la cual se encarga de crear la aplicación que se desea montar en el sistema (línea 7),
- Crear Sistema: El sistema crea por medio de la clase VRJPFSystem (la adaptación del framework para el sistema VRJUGGLER + PERFORMER + VTK) un nuevo sistema y lo configura (línea 11), as4ociándole la aplicación creada (línea 15).
- Ejecutar: Por ultimo ya configurado todo el sistema, entra a correr la aplicación por medio de instrucción execute (línea 18).

Como se ha mostrado anteriormente se configuró todo el framework InTml para poder ser utilizado en la infraestructura propuesta, sin embargo, es claro que primero debe configurarse la infraestructura, por lo cual en la siguiente sección se mostrará como se integraron cada una da las librerías utilizadas.

### 3.2.1. Integración de Librerías

Como se pudo ver en la Imagen 5, en la actualidad se están utilizando las siguientes librerías, Framework InTml,.VrJuggler, VTK, Performer y Vrpn. A continuación se mostrará la integración de cada una de las librerías al sistema.

**Framework InTml + VrJuggler**

La forma en que VrJuggler ejecuta sus aplicaciones es por medio del registro de éstas a su kernel, por medio de dos de sus métodos, el método start y el método setApplication, al integrarlo con el sistema esto no se modifica, lo único que se hace es cambiarlo de posición, ya que estos llamados se hacen en el método setApp (Tabla 3) de la clase VRJPFSystem.

**Tabla 3 Método SetApp de la Clase VRJPFSystem (VRJPFSystem.cxx Línea 201)**

```
1  void VRJPFSystem::setApp( VRJPFApp* a )
2  {
3    InTmlSystem::setApp( a );
4    kernel->start();
5    kernel->setApplication(a);
6  }
```

## VrJuggler + VRPN

Para poder integrar estas dos librerías se trabajó directamente con los archivos de configuración de VrJuggler, ya que por medio de estos se permite la comunicación con un servidor VRPN por medio de una red TCP-UDP/IP en un puerto específico.

Como se comentó en la primera sección de este documento VRPN identifica sus dispositivos por medio de un nombre significativo, por lo cual para poder obtener la información desde un acceso remoto es necesario utilizar la siguiente cadena de conexión "device@type://machine:port", donde "device" es el identificador del dispositivo, "type" es el tipo de conexión (TCP O UDP) y "port" es el puerto en donde está corriendo el servidor, un ejemplo particular es el acceso a los trackers en nuestro laboratorio, para obtener los datos de los dispositivos se debe utilizar la siguiente línea de conexión "Tracker0@girardot.uniandes.edu.co:3883".

El archivo de configuración de VrJuggler que se necesita para soportar la integración con VRPN deberá contener la información que se muestra en la siguiente tabla, esto se logra utilizando la aplicación VRJConfig, que permite hacerlo de una manera mas amigable.

**Tabla 4 Archivo XML de Configuración para VrJuggler habilitando VRPN**

```
1   <vrpn name="Vrpn" version="1">
2   <tracker_server>Tracker0@servidor</tracker_server>
3   <button_server>Button0@servidor</button_server>
4     <tracker_count>2</tracker_count>
```

```
5    <button_count>4</button_count>
6    <position_filters>
7      <position_transform_filter name="Position Filters" version="1">
8        <pre_translate>0.0</pre_translate>
9        <pre_translate>0.42672</pre_translate>
10         <pre_translate>0.0</pre_translate>
11         <pre_rotation>90.0</pre_rotation>
12         <pre_rotation>0.0</pre_rotation>
13         <pre_rotation>0.0</pre_rotation>
14         <custom_scale>1.0</custom_scale>
15         <device_units>1.0</device_units>
16         <post_translate>0.0</post_translate>
17         <post_translate>0.0</post_translate>
18         <post_translate>0.0</post_translate>
20         <post_rotation>0.0</post_rotation>
21         <post_rotation>0.0</post_rotation>
22         <post_rotation>0.0</post_rotation>
23       </position_transform_filter>
24     </position_filters>
25   </vrpn>
```

Como se puede observar se puede configurar un servidor de tracker y otro de botones (líneas 2 y 3), configurados con la cadena de conexión especificada anteriormente, las líneas 4 y 5 indican la cantidad de dispositivos registrados con esa identificación.

Adicionalmente a esto es necesario tener habilitado el administrador de entradas, el cual permite que VrJuggler conozca el comportamiento del dispositivo instalado, en este caso el servidor VRPN. A continuación se muestra un ejemplo de cómo habilitar éste en el archivo de configuración:

**Tabla 5 Archivo XML de Configuración para Habilitar el administrador de Entradas**

```
1  <input_manager name="Input Manager" version="2">
2    <driver_path>
3      ${VJ_BASE_DIR}/lib/gadgeteer/drivers
4    </driver_path>
5    <driver_path>
6      ${VJ_BASE_DIR}/lib32/gadgeteer/drivers
7    </driver_path>
8    <driver>VRPN_drv</driver>
9  </input_manager>
```

La variable ${VJ_BASE_DIR} deberá contener el directorio en donde queda instalado VrJuggler.

**VrJuggler + Performer**

Una aplicación normal en VrJuggler tiene que heredar de una clase genérica llamada App del paquete vrj (vrj::app) que suministra la distribución. Para poder soportar el grafo de escena generado por Performer se tiene que modificar la herencia de la clase, ya que en este momento la clase hereda de otra conocida como PfApp, un ejemplo de este cambio sería el siguiente:

```
class VRJPFApp: public vrj::PfApp
```

Con esto ya se puede crear la escena en Performer generando el nodo principal por medio de un objeto pfGroup y añadir todos los hijos que éste vaya a tener, por ejemplo, en el caso del visualizador de objetos Performer, que se mencionará mas adelante, los hijos se agregan de la siguiente manera:

```
rootNode = new pfGroup;
Obj = (<PerformerObject*>(*indexObject))->getGeom();
rootNode->addChild(Obj);
```

**VTK + Performer**

VTK es una herramienta muy poderosa para el desarrollo de aplicaciones con procesamiento de imágenes, pero su base está desarrollada en OPENGL y no existe una manera directa de pasar los objetos generados en ésta a Performer, por lo tanto es necesario utilizar una clase adicional conocida como VtkActorToPF. Todas la aplicaciones desarrolladas en VTK, constan de tres pasos básicos, primero generar el modelo 2D o 3D (línea 1 a 7), segundo asociarle un mapper (línea 9) y por último asociar el mapper con el

actor (línea 10), un ejemplo de esto se puede observar en el archivo mace.cxx (Tabla 6) de la distribución 3.2 el cual genera una esfera con espinas.

**Tabla 6 Ejemplo de VTK mace.cxx**

```
1  vtkSphereSource *sphere = vtkSphereSource::New();
2  vtkGlyph3D *glyph = vtkGlyph3D::New();
3     glyph->SetInput(sphere->GetOutput());
4     glyph->SetSource(cone->GetOutput());
5     glyph->SetVectorModeToUseNormal();
6     glyph->SetScaleModeToScaleByVector();
7     glyph->SetScaleFactor(0.25);
8    vtkPolyDataMapper *spikeMapper = vtkPolyDataMapper::New();
9     spikeMapper->SetInput(glyph->GetOutput());
10  vtkActor *spikeActor = vtkActor::New();
11  spikeActor->SetMapper(spikeMapper);
```

El actor de VTK es el que permite visualizar el modelo en si, por lo cual nos apoyamos en VtkActorToPF para poder transformar éste a un nodo Performer, a continuación se muestra un ejemplo de cómo agregarlo al grafo por medio del método GetGeode (línea 7).

**Tabla 7 Uso de VtkActorToPF**

```
1  skin = vtkActor::New();
2  skin->SetMapper(skinMapper);
3  vtkActorToPFTranslator *translator
4    translator = vtkActorToPFTranslator::New();
5    translator->SetActor(skin);
6    translator->Init();
7  rootNode->addChild(translator->GetGeode());
```

Todas estas librerías permiten que el sistema funcione, pero se debe seguir una serie de procedimientos para poder usar esta infraestructura adecuadamente, los cuales son explicados en la siguiente sección.

## 3.3. Procedimientos

A continuación se describen los procesos involucrados para el funcionamiento y uso de la infraestructura propuesta, están divididos en tres grandes áreas, configuración, desarrollo y la generación de aplicaciones.

### 3.3.1. Configuración (Conexión HW y Calibración)

Se puede dividir la configuración del sistema en 3 partes, dispositivos, software y ejecución.

**Dispositivos**

El primer proceso es el de conexión y calibración de dispositivos. Dado que se desea crear un servidor de dispositivos el primer paso es la instalación de VRPN, éste se puede montar tanto en un servidor Windows como en un servidor Linux, en el caso del laboratorio se tienen una instalación en ambas plataformas, cada una con un grupo de dispositivos diferentes.

Este proceso se puede dividir en 5 partes fundamentales, los cuales se describirán a continuación:

- Instalación del software: Se debe instalar la versión de VRPN correspondiente al sistema operativo sobre el cual se montara el servidor, en caso de se Linux se compila con gcc en caso de Windows con visual c++.
- Conexión de dispositivos: después de el primer paso se procede a la instalacion física de los dispositivos en el servidor, ya sea en los puertos seriales, los usb u otros, se debe instalar tanto los cables de datos, como los cables de corriente, un ejemplo de instalación se puede observar en la Imagen 13.

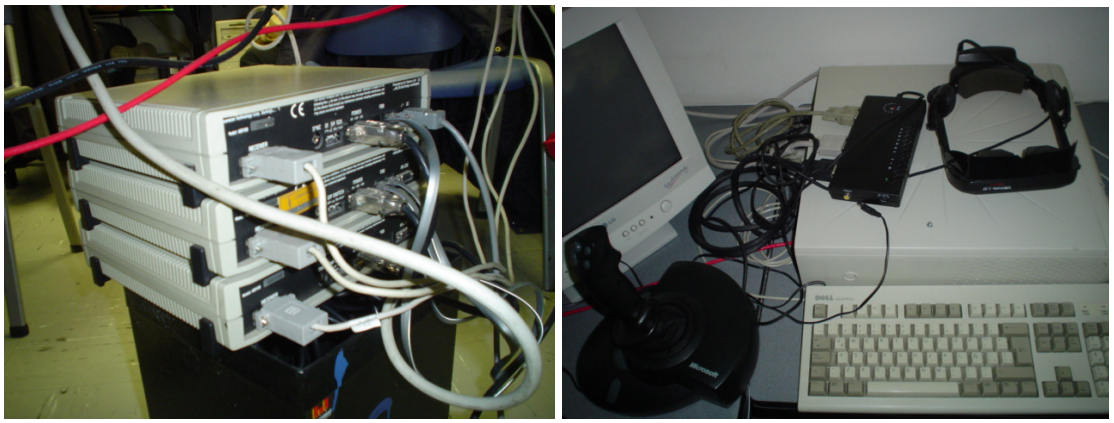

**Imagen 13 Instalación de los Dispositivos**

- Configuración de los dispositivos: Los dispositivos deben ser registrados en el sistema por medio del archivo de configuración vrpn.conf el cual se encuentra en la carpeta de instalación. En la siguiente tabla (Tabla 8) se muestran los cambios en el archivo para cada uno de los dispositivos del laboratorio.

| | | | | | | |
|---|---|---|---|---|---|---|
| **Flock of Birds (Linux)** | | | | | | |
| vrpn_Tracker_Flock | Tracker0 | 3 | /dev/ttyS0 | 38400 | 1 | N |
| **Joystick (Windows)** | | | | | | |
| vrpn_DirectXFFJoystick | Joystick0 | 60 | 200 | | | |
| **Glove (Windows)** | | | | | | |
| vrpn_5dt | glove_5dt | COM1 | 19200 | 1 | 0 | |
| **Gamepad (Windows)** | | | | | | |
| vrpn_Joywin32 | gamepad | 2 | 60 | 0 | 35 | |

**Tabla 8 Configuración de Dispositivos**

- Probar los dispositivos: Se debe correr los aplicativos de prueba que permiten probar la efectiva configuración de los dispositivos, primero debe ejecutarse el servidor (vrpn_server) y luego un cliente que muestre los valores por pantalla

(vrpn_print_devices), si los valores que son mostrados en la pantalla tienen relación con los dispositivos, estos han quedado bien configurados.

- Calibración de los dispositivos: este proceso se debería desarrollar antes de la ejecución de una aplicación y periódicamente para corregir cambios en el ambiente. En el caso de los trackers para el laboratorio se construyó una mesa con marcas de calibración cada 10cm, y la posibilidad de variar su altura, para tomar datos del tracker en una grilla 3D de 70cm de lado (Imagen 14). Estos datos se pasan a un archivo de configuración que es leído por VrJuggler logrando así que los datos obtenidos tengan una relación directa con el mundo.

**Imagen 14 Mesa para calibración de los Trackers**

## Software

A nivel de software se pueden modificar los dispositivos de entrada y salida los cuales están representados como filtros. Hacer un cambio de dispositivos es hacer un cambio de filtros, buscar el filtro del dispositivo que se quiere, utilizarlo y conectarlo de una manera adecuada a los demás filtros en la aplicación. Al igual que con los dispositivos hacer un cambio en las técnicas de interacción se logra con un simple cambio de filtro, es por esto que entre mayor sea la cantidad de filtros desarrollados es mayor la cantidad de elementos disponibles para el desarrollo de aplicaciones.

Para el desarrollo de un nuevo dispositivo en el sistema se deben crear los archivos de configuración para VrJuggler necesarios para leer los dispositivos desde VRPN, dándoles un nombre significativo que les permita ser registrados en el sistema. Para poder configurarlo en InTml, se deben crear los archivos necesarios para cada uno de los ambientes con los que se quiere trabajar.

**Ejecución**

Por último, consideramos un procedimiento para la ejecución de la aplicación. Actualmente se corre como una aplicación convencional que recibe los parámetros por línea de comandos, estos modifican la aplicación de acuerdo a sus valores. La idea final es correr la aplicación por medio de ant[9], lo que dividiría el proceso en dos, el primero un pequeño programa que lee parámetros en línea y cambia el archivo de propiedades, y después se correría ant con el archivo creado.

En la actualidad los parámetros que se reciben son para cambios de rendimiento y opciones de alto nivel, dentro de los que se encuentran la calidad de la imagen y la visualización en estéreo. Los parámetros de alto nivel se reflejan en el tratamiento interno de los filtros que representan objetos gráficos. En la implementación actual de filtros que representan objetos, se maneja de dos maneras completamente distintas: en Performer se mantiene un estándar de nombres y en VTK se mantiene un parámetro dentro del filtro que se crea al empezar la aplicación.

### 3.3.2. Desarrollo

Como se ha indicado anteriormente la arquitectura InTml propone el uso de filtros y puertos en todos los componentes de la aplicación, es por eso que cualquiera de los elementos a crear va a estar ligado de alguna manera con una clase genérica (Filter), la

[9] http://ant.apache.org/

cual establece los requerimientos mínimos que un filtro debe tener, dentro de los cuales se encuentran tres métodos, setup, createPorts y execute.

El primer de ellos como su nombre lo indica es la configuración, normalmente se le da el nombre que se le ha asignado al filtro; el segundo configura los puertos del filtro, tanto los de entrada como los de salida y deben inicializarse en este momento; y por último el método execute, el cual obtiene la información de los puertos de entrada, la modifica y envía la nueva información por los puertos de salida.

De acuerdo al tipo de filtro que se desee crear existe una forma básica para unirlo al sistema, por lo tanto a continuación se mostrarán las dependencias básicas de las tres modalidades de filtros.

**Interacción**

Un filtro de interacción es un filtro lógico, siempre tendrá que decidir cuando actuar y en caso de hacerlo obtener un cálculo y pasarlo al siguiente filtro conectado. Para tomar las decisiones el filtro necesita información de entrada y calculará información de salida, esto se ve reflejado en diferentes puertos, tanto en el de entrada como en el de salida, en la gráfica siguiente se ve reflejado en los Iport y en los OPort. La clase a crear deberá heredar directamente de la clase Int, ésta tiene como objetivo establecer la configuración básica de un filtro intermedio, a continuación se mostrará el diagrama de clases para el dispositivo Key2RGB, mostrado más adelante.

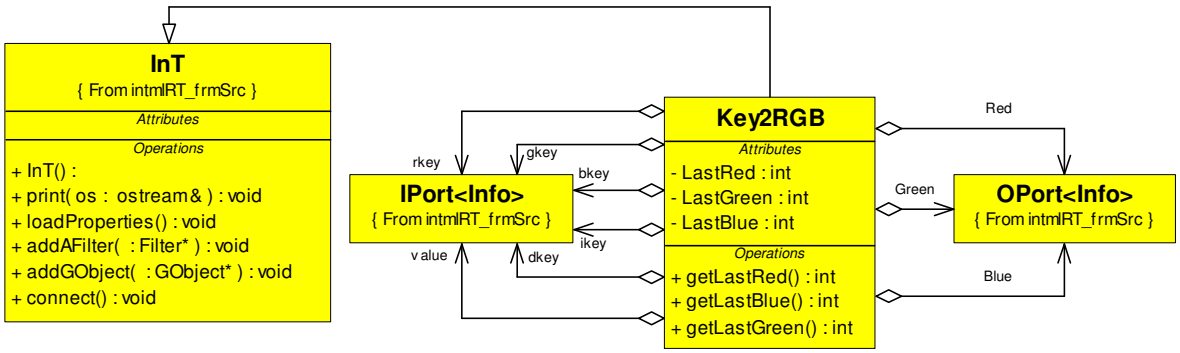


**Imagen 15 Descripción de un nuevo filtro de Interacción (Key2RGB)**

Adicionalmente se pueden establecer atributos que permitan obtener información de los elementos que han estado pasando por el filtro o para mantener alguna información de retroalimentación.

## Dispositivos

A diferencia de los filtros para interacción, los filtros de dispositivos no tienen la necesidad de tener puertos de entrada, ya que están representando un dispositivo externo, el cual debería se manipulado por el usuario y por lo tanto mantendría enviando información constantemente, tal como se observa en la siguiente gráfica para el filtro que representa un teclado (Imagen 16).

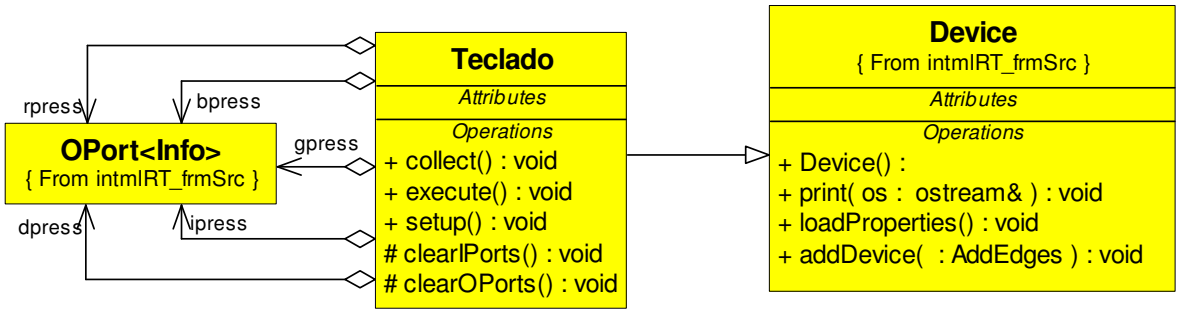


**Imagen 16 Descripción de un nuevo filtro de dispositivos**

## Objetos

Los objetos tienen una representación muy similar a la de los dispositivos, como se puede ver en la siguiente gráfica (Imagen 17), la cual representa un cubo creado en Performer.

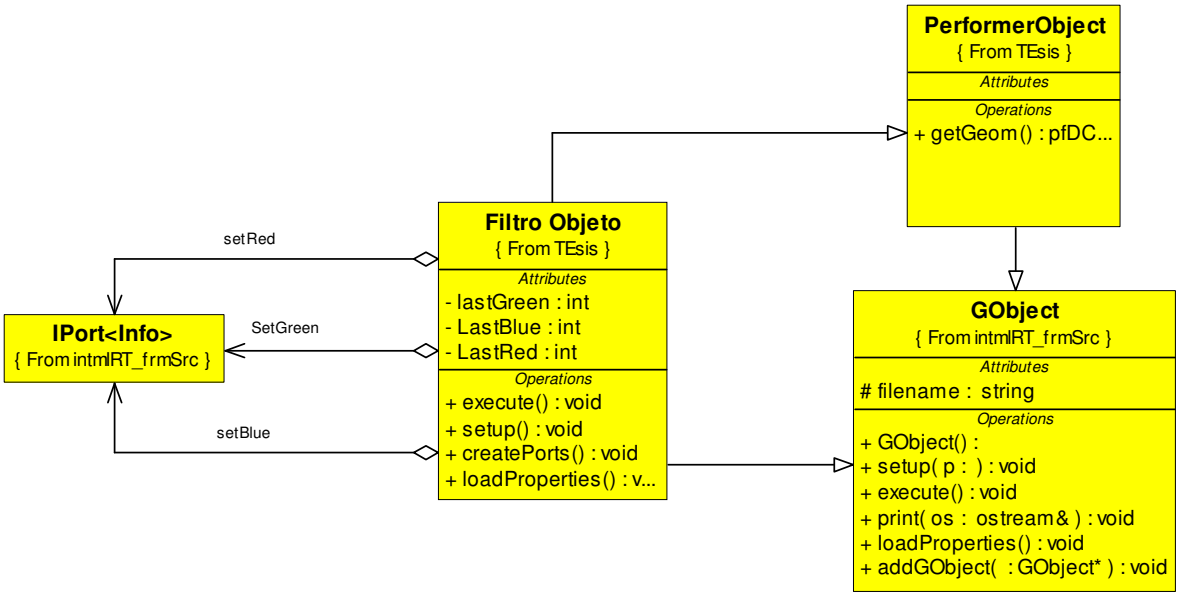


**Imagen 17 Descripción de un nuevo filtro de visualización**

Los filtros que representan objetos, se llegaría a pensar que solo tienen puertos de entrada, pero en algunas ocasiones estos filtros deben informar al sistema de algún cambio en el modelo, por lo cual estos deben tener filtros de entrada y de salida.

### 3.3.3. Demos

A continuación se mostrará el procedimiento para crear algunas aplicaciones simples en el sistema, empezando por la creación de una aplicación y después como modificarla para crear otra nueva aplicación.

**Crear una aplicación**

Para crear una nueva aplicación en el sistema es necesario tener claro los tres tipos de componentes que interactúan con una aplicación VR (dispositivos, técnicas de interacción y objetos de visualización) así como las entradas y salidas que cada uno de estos va a necesitar. Con esto claro comienza el diseño de la aplicación, el cual consiste en encontrar la relación entre cada uno de los elementos.

Un ejemplo sencillo con fines ilustrativos, sería crear una aplicación que muestre un cubo y permita cambiar su tonalidad de color. Lo primero que se debe analizar son los elementos a tener en cuenta en esta aplicación.

En relación a los dispositivos deseamos manipular la aplicación por medio del teclado, vamos a utilizar teclas con algún significado, por lo cual se utilizaran las teclas 'R', 'G', 'B', 'D', 'I' representando los respectivos colores rojo, verde y azul y si se está decrementando o incrementando. Al hacer esto implícitamente estamos definiendo una técnica de interacción ya que estamos dando una relación entre el color y cada una de las diferentes teclas, oprimir una tecla conlleva a un cambio en el color RGB. Por último necesitamos el objeto a visualizar y como se mencionó anteriormente este será un cubo.

Haciendo un recuento tenemos los siguientes elementos en nuestra aplicación:

- Cube (visualización).
- Teclado (dispositivo).
- Key2RGB (interacción).

Pero aun no hemos definido las entradas y salidas que cada uno de ellos necesita para poder interactuar con los demás, por lo tanto necesitamos empezar a analizar cada uno de los elementos y comenzaremos en orden.

Cube: En este momento vemos que el objeto necesita recibir de alguna manera el color que indica su nueva tonalidad, por lo tanto necesitamos representarlo de alguna manera para poder definir las entradas que necesita el objeto. Ya que sabemos que el color que nos llegará está en formato RGB, podemos definirlo estableciendo tres entradas para cada uno de los colores.

Podemos agregar algunos elementos de salida como el valor RGB actual, otros filtros en la aplicación podrían necesitarlo, pero en este caso no son necesarios. El elemento que necesitamos crear tienen la siguiente estructura (Imagen 18).

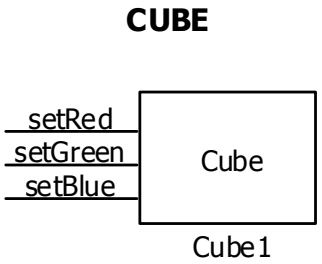


**Imagen 18 Filtro de Visualización Cube**

Teclado: Al contrario que los filtros de visualización los dispositivos tienen implícitas salidas, ya que estos son la fuente de información del usuario hacia el sistema, por lo tanto la representación sería muy sencilla y es la siguiente (Imagen 19):

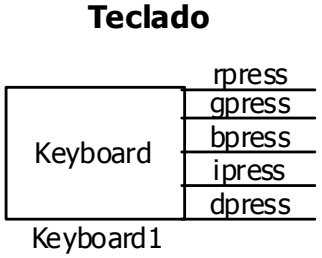


**Imagen 19 Filtro para un dispositivo**

Key2RGB: El último elemento es el filtro de interacción, ya que tenemos la visualización y los dispositivos, necesitamos algún medio de comunicación entre estos, y este medio es la

interacción, por lo tanto este filtro a diferencia de los anteriores tendrá entradas y salidas. Las entradas estarán ligadas a los dispositivos y las salidas a los objetos, en este caso en particular el teclado tiene cinco salidas y el objeto tres entradas, adicionalmente tenemos que establecer en cuanto se incrementara el color. Por lo cual la representación de nuestro filtro sería la siguiente (Imagen 20).

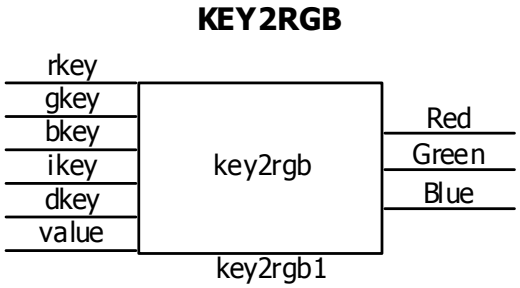


**Imagen 20 Filtro de Interacción**

Ya que se han definido cada uno de los filtros el proceso a seguir consiste en conectar cada uno de los elementos y así crear la aplicación. Para el anterior ejemplo la aplicación sería la siguiente (Imagen 21):

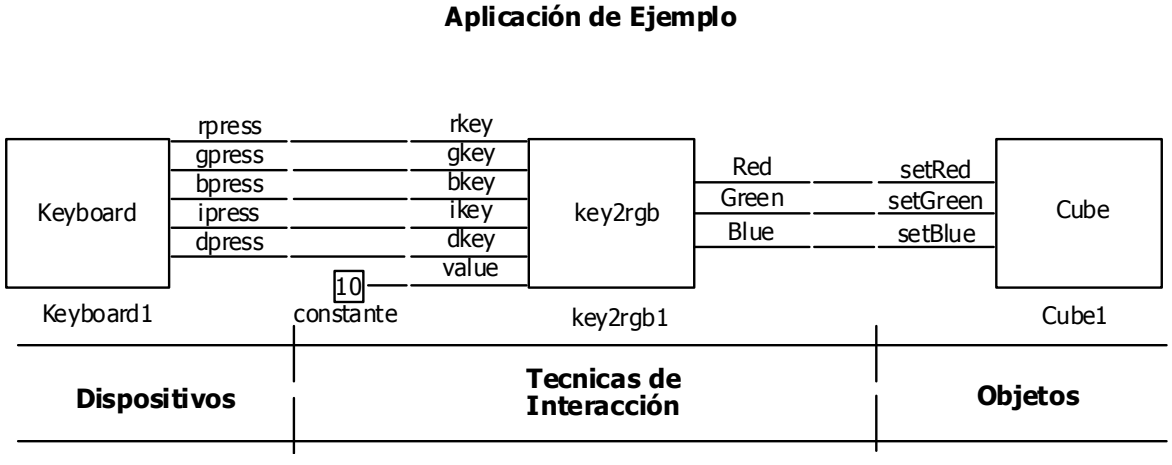


**Imagen 21 Aplicación Ejemplo**

El archivo InTml que representa la anterior aplicación es el siguiente (Tabla 9):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE App PUBLIC "-//pfiguero//3D Interaction Techniques ML//EN"
        "http://localhost/~pfiguero/InTmlTemp/spec/InTml.dtd">
<App id="example1">
        <ShortDesc>Aplicacion de ejemplo  1</ShortDesc>
        <Import id="example"/>
        <IDevice id="keyboard1" type="Keyboard"/>
        <Object id="cube1" type="Cube"/>
        <Filter id="key2rgb1" type="Key2RGB"/>
       <Constant id="constante" type="int" value="10"/>
        <Binding iE="keyboard1" iP="rpress" oE="key2rgb1" oP="rkey"/>
        <Binding iE="keyboard1" iP="gpress" oE="key2rgb1" oP="gkey"/>
        <Binding iE="keyboard1" iP="bpress" oE="key2rgb1" oP="bkey"/>
        <Binding iE="keyboard1" iP="ipress" oE="key2rgb1" oP="ikey"/>
        <Binding iE="keyboard1" iP="dpress" oE="key2rgb1" oP="dkey"/>
        <Binding iE="_self" iP="constante" oE="key2rgb1" oP="value"/>
        <Binding iE=" key2rgb1" iP="Red" oE="cube1" oP="setRed"/>
        <Binding iE=" key2rgb1" iP="Green" oE="cube1" oP="setGreen"/>
        <Binding iE=" key2rgb1" iP="Blue" oE="cube1" oP="setBlue"/>
</App>
```

**Tabla 9 Archivo InTml**

## Modificar Aplicaciones

Uno de los mayores beneficios de la infraestructura es que hacer un cambio en la aplicación no requiere de grandes cambios de software sino de estructura en los filtros, por lo tanto se pueden hacer cambios en los tres tipos de elementos involucrados, a continuación se mostrarán algunos de los cambios posibles y ejemplos que los ilustran.

## Técnicas de Interacción

Si se desea que la aplicación del ejemplo anterior no interactué con el teclado de la manera como se está haciendo, se debería cambiar la técnica de interacción actual. Existen diferentes soluciones para esto, por ejemplo:

- Modificar la cantidad de teclas predefinidas, no sólo se utiliza ‘R’,’G’,’B’, para cambiar el color sino que además se utiliza ‘E’, ’F’, ’V’, unas incrementan los canales, las otras lo decrementan, en este caso las teclas ‘I’ y ‘D’, dejan de utilizarse.
- Agregar otro dispositivo, como el Mouse y modificar el filtro de interacción para que ahora reciba los eventos (clic derecho, centro e izquierdo) y la posición (x, y).

Si se desea utilizar la primera solución, el comportamiento del filtro de interacción cambia completamente, por lo cual es necesario crear otro filtro que se comporte con la nueva interacción. El filtro anterior actuaba de acuerdo a lo recibido en sus entradas y modificaba el color de acuerdo a los valores de sus puertos ikey y dkey, el nuevo filtro solo debe mirar las entradas e incrementar o decrementar el valor sin ningún condicional, a continuación se muestra el resultado de utilizar este filtro en la aplicación (Imagen 22).

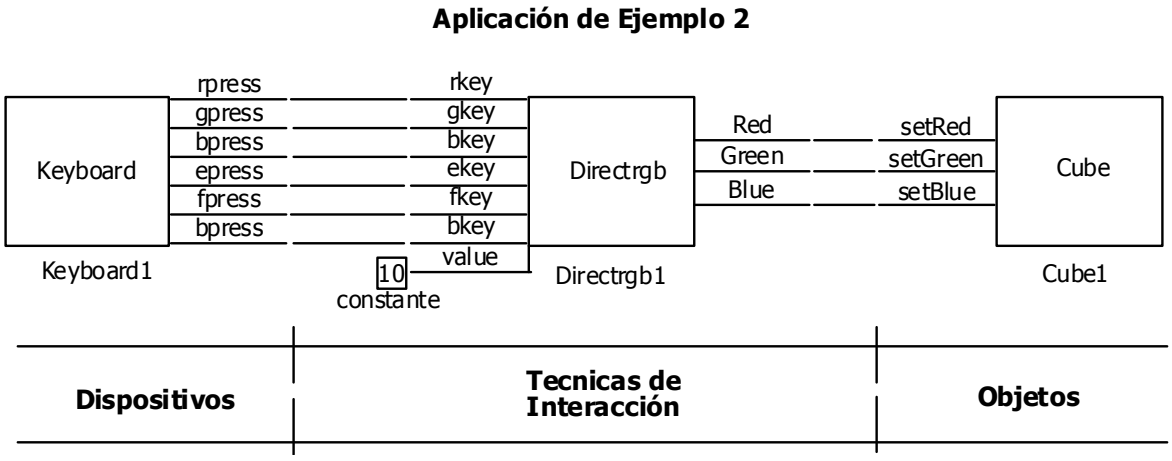


**Imagen 22 Aplicación de Ejemplo Modificada con interacción 1**

Para el segundo caso se tienen que tener en cuenta cada uno de los elementos anteriormente descritos, pero adicionalmente el filtro cambia completamente su estructura (Imagen 23), en este caso el funcionamiento del filtro sería el siguiente: hay puertos de entrada distintos para los distintos eventos (startRed, stopRed, startBlue, ...) los cuales están en relación directa con los botones del Mouse, así right-button es Red, center-button es Green y left-button es Blue, la intensidad de color es dada por la posición relativa en la pantalla de la posición (x,y) del Mouse.

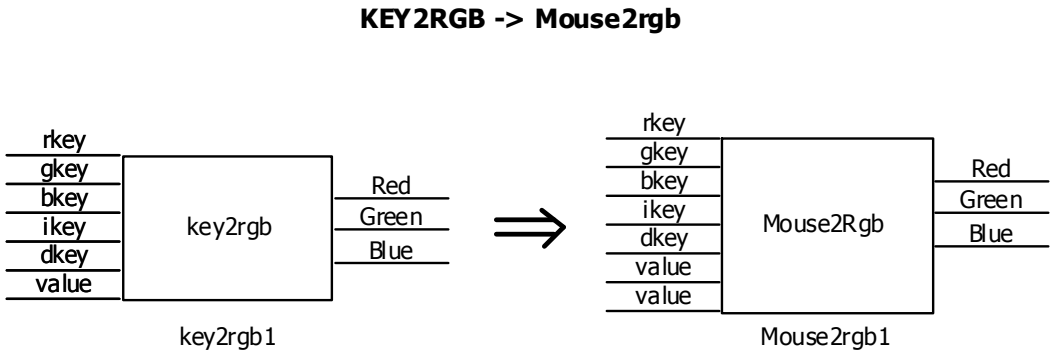


**Imagen 23 Cambio en Filtro de Interacción**

## Dispositivos

Existen cambios que requieren el uso de diferentes dispositivos y para hacer esto se necesita cambiar el filtro respectivo de acuerdo a los filtros desarrollados en el sistema, en el caso del ejemplo anterior la creación del dispositivo y la unión con la aplicación seria la siguiente (Imagen 24).

.

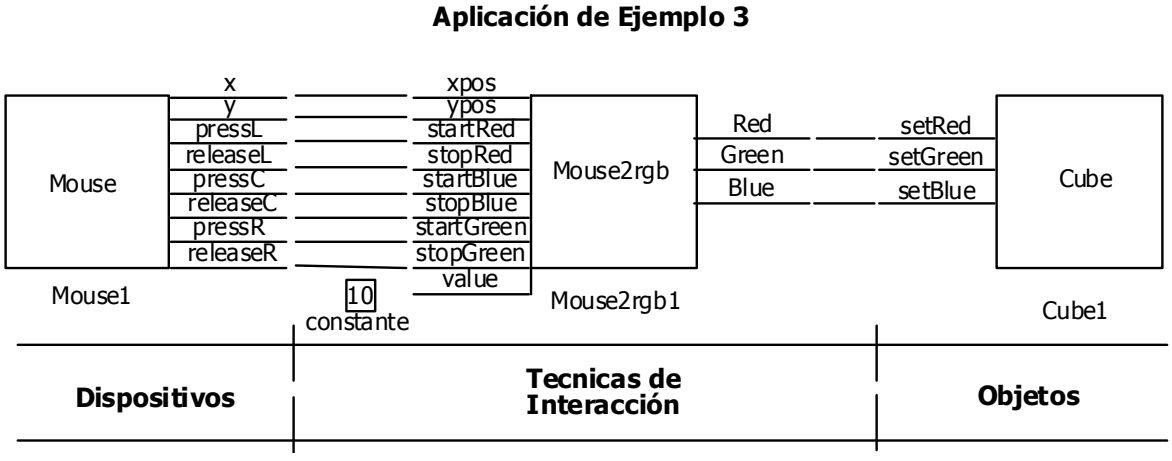


**Imagen 24 Aplicación de Ejemplo Modificada con interacción 2**

Esta configuración de la aplicación muestra que también es posible emular los dispositivos, si lo que queremos lograr es que la aplicación funcione con un gamepad, lo único que se debería hacer es crear el filtro que representa este dispositivo, con la configuración que se muestra a continuación (Imagen 25).

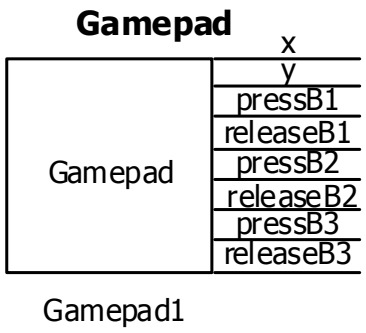


**Imagen 25 Filtro representativo de un Gamepad**

Para conectarlo a la aplicación se debería conectar con el filtro de interacción y sin ningún proceso adicional se lograría crear la aplicación como se muestra a continuación (Imagen 26).

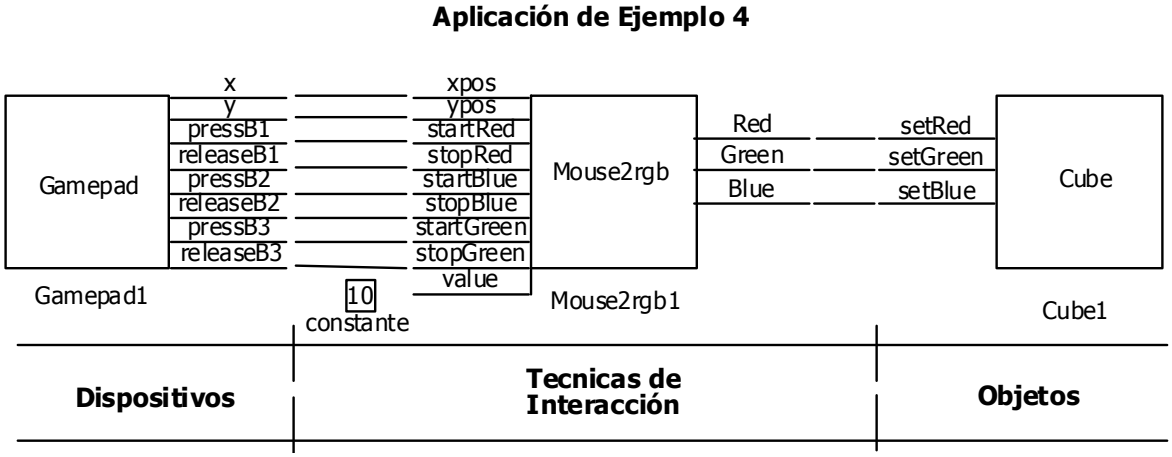


**Imagen 26 Aplicación de Ejemplo Modificada para el Gamepad**

## Objetos

Estos últimos no tienen un cambio drástico, ya que normalmente son independientes, por ejemplo, si ya no se desea visualizar el cubo sino un objeto mucho más complejo como un archivo de Performer se tendría que cargar el objeto y conectarlo con los filtros necesarios para modificarlo.

En el siguiente capítulo se mostrarán tres aplicaciones que muestran con mayor profundidad tanto el uso de los filtros como el cambio de filtros para generar otra aplicación.

# 4. EJEMPLOS

Para validar el modelo que se propuso se desarrollaron dos aplicaciones, que permiten ver el funcionamiento del ambiente de trabajo. Estas aplicaciones se mostrarán a continuación, con una breve descripción de su desarrollo y como fueron reutilizadas para transformar diferentes aplicaciones en diferentes ambientes.

## 4.1. Visualizador Performer

**Imagen 27 Imágenes de la Aplicación Visualizador Performer**

El objetivo de esta aplicación es dar las opciones típicas de un visualizador (zoom y rotación), para cualquier tipo de objeto Performer. la gráfica siguiente (Imagen 28) muestra el diseño de la aplicación InTml.

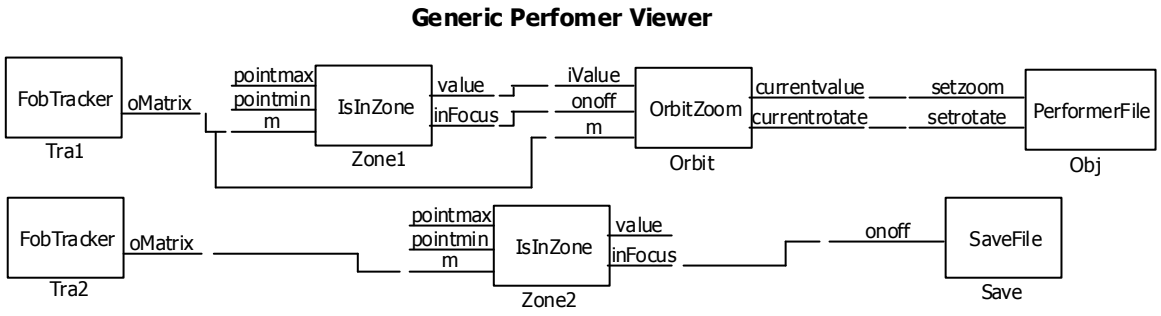


**Imagen 28 Visualizador de Objetos Performer**

Esta aplicación requirió el desarrollo de cuatro tipos de filtros, uno de dispositivos, uno para objetos y dos para interacción, la descripción de estos filtros se hará a continuación mostrando diferentes diagramas y explicando parte de su código.

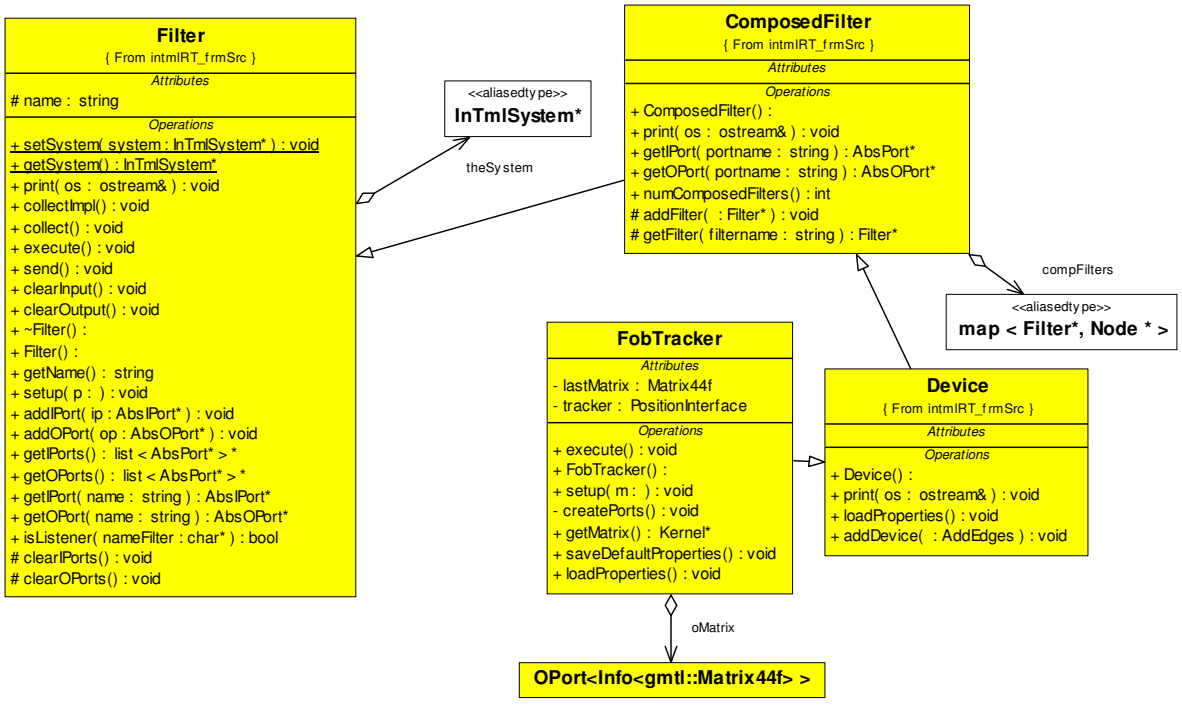


**Imagen 29 Diagrama de Clases Filtro FobTracker**

- Fobtracker: representa un tracker FOB, el cual retorna la matriz de transformación del punto, representada como una matrix44f de gmtl[10].

[10] gmtl es un API incluido con VrJuggler para trabajar las operaciones vectoriales y matriciales.

Como se puede observar este dispositivo tiene un solo puerto de salida llamado oMatrix y ninguno de entrada. Para poder comunicarse con los demás filtros se le debe especificar un nombre a cada uno, en este caso "omatrix" es llamado "matrix" como se ve a continuación (Tabla 10):

**Tabla 10 Creación de los Puertos en el Filtro FobTracker (FobTracker.cxx Línea 44)**

```
void FobTracker::createPorts()
{
  // Add OPort of type gmtl::Matrix44f
  oMatrix=new OPort<Info<gmtl::Matrix44f> >(this,"matrix");
  oports.add( oMatrix );
};
```

La forma como éste envía la información a los puertos de salida es por medio del método execute(), el cual debe encargarse del procesamiento de la información, en este caso en particular obtiene la información del dispositivo que está representando y envía la información por medio de su puerto de salida, como se ve a continuación (Tabla 11).

**Tabla 11 Método Execute de FobTracker (FobTracker.cxx Línea 24)**

```
void FobTracker::execute()
{
 Device::execute();
 gmtl::Matrix44f* m=new gmtl::Matrix44f(tracker->getData());
 Info<gmtl::Matrix44f>*info=new Info<gmtl::Matrix44f>(m);
 info->addRef();
 oMatrix->push( info );
}
```

- IsInZone: filtro lógico que calcula si el punto que llega al puerto m, se encuentra dentro de la región delimitada entre pointmin y pointmax, retornando un valor verdadero (inFocus) y la correspondiente posición normalizada en el espacio (value).

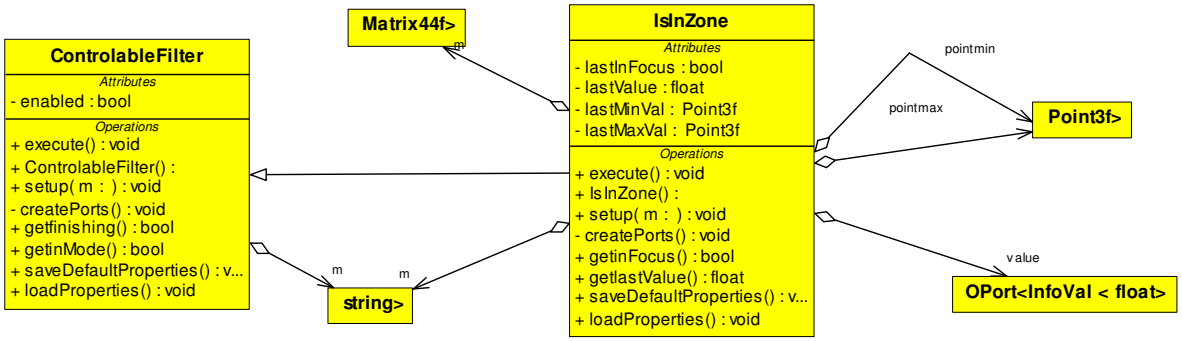


**Imagen 30 Diagrama de Clases Filtro IsInZone**[11]

Como en el anterior filtro en éste se analizarán los dos métodos principales, la creación de los puertos tanto de entrada como de salida y su método de ejecución.

**Tabla 12 Metodo CreatePorts de IsInZone (IsInZone.cxx Línea 101)**

```
void IsInZone::createPorts()
{
    pointmax = new IPort<Info<gmtl::Point3f> >( this, "pointmax" );
    iports.add( pointmax );
    m = new IPort<Info< gmtl::Matrix44f> >( this, "m" );
    iports.add( m );
    pointmin = new IPort<Info<gmtl::Point3f> >( this, "pointmin" );
    iports.add( pointmin );
    inFocus =  new OPort<InfoVal<bool> >( this, "inFocus" );
    oports.add( inFocus );
    value =  new OPort<InfoVal<float> >( this, "value" );
    oports.add( value );
}
```

Como se observa en el método anterior este filtro está inicializando tres puertos de entrada (pointmax, pointmin y m) y dos de salida (value y inFocus), los cuales reciben el mismo nombre que la variable que los contiene.

**Tabla 13 Metodo execute de IsInZone (IsInZone.cxx Línea 24)**

```
void IsInZone::execute()
{
```

[11] ControlableFilter tiene como función deshabilitar o no un filtro

```
  ControlableFilter::execute();
  if( pointmax->hasInfo() ){
    Info<gmtl::Point3f> *info = pointmax->popTyped();
    lastMaxVal = *info->getInfo();
    info->delRef();
  } if( pointmin->hasInfo() ) {
    Info<gmtl::Point3f> *info = pointmin->popTyped();
    lastMinVal = *info->getInfo();
    info->delRef();
  } if( m->hasInfo() ) {
    Info<gmtl::Matrix44f>* info = m->popTyped();
    gmtl::Point3f p;
    gmtl::setTrans( p, *info->getInfo() );
    if ( p(0)>=lastMinVal(0) && p(1)>=lastMinVal(1) && p(2)>=lastMinVal(2) &&
         p(0)<=lastMaxVal(0) && p(1)<=lastMaxVal(1) && p(2)<=lastMaxVal(2) ){
      InfoVal<bool>* info = new InfoVal<bool>( true );
      info->addRef();
      inFocus->push( info );
      float temp=0, dist1, dist2;
      dist1  = ( lastMaxVal(0) - lastMinVal(0) ) * ( lastMaxVal(0) - lastMinVal(0) );
      dist1 += ( lastMaxVal(1) - lastMinVal(1) ) * ( lastMaxVal(1) - lastMinVal(1) );
      dist1 += ( lastMaxVal(2) - lastMinVal(2) ) * ( lastMaxVal(2) - lastMinVal(2) );

      dist2  = ( lastMaxVal(0) - lastMinVal(0) ) * ( p(0) - lastMinVal(0) );
      dist2 += ( lastMaxVal(1) - lastMinVal(1) ) * ( p(1) - lastMinVal(1) );
      dist2 += ( lastMaxVal(2) - lastMinVal(2) ) * ( p(2) - lastMinVal(2) );
      temp =  dist2 / dist1;

       InfoVal<float>* info2 = new InfoVal<float>( temp );
       info2->addRef();
       value->push( info2 );
    } else {
      InfoVal<bool>* info = new InfoVal<bool>( false );
      info->addRef();
      inFocus->push( info );
    }
    info->delRef();
  }
}
```

El comportamiento de este filtro está basado en la proyección de vectores, por lo cual como se observa, calcula el producto punto entre el vector producido por los límites de la zona, y el vector generado por el valor mínimo y el valor del punto ‘m’; al igual calcula el producto punto entre si mismo para poder encontrar la proyección (Imagen 31).

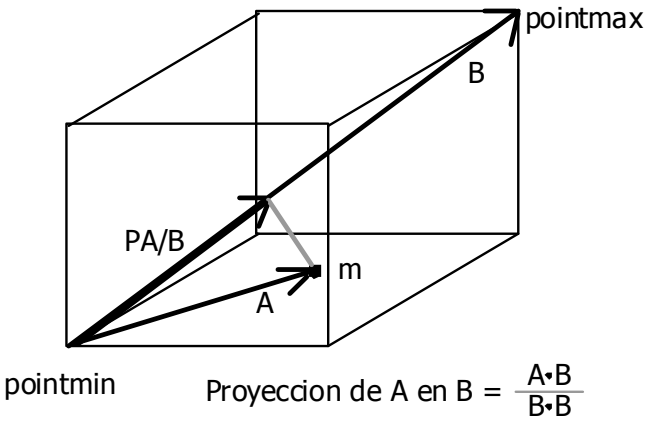


**Imagen 31 Comportamiento basado en proyección de vectores**

- OrbitZoom: filtro lógico que calcula el zoom y la rotación si se encuentra activado (onoff). El valor de la translación es calculado por el valor recibido en iValue y la rotación por la matriz, los correspondientes valores son enviados a currentvalue y currentrotate.

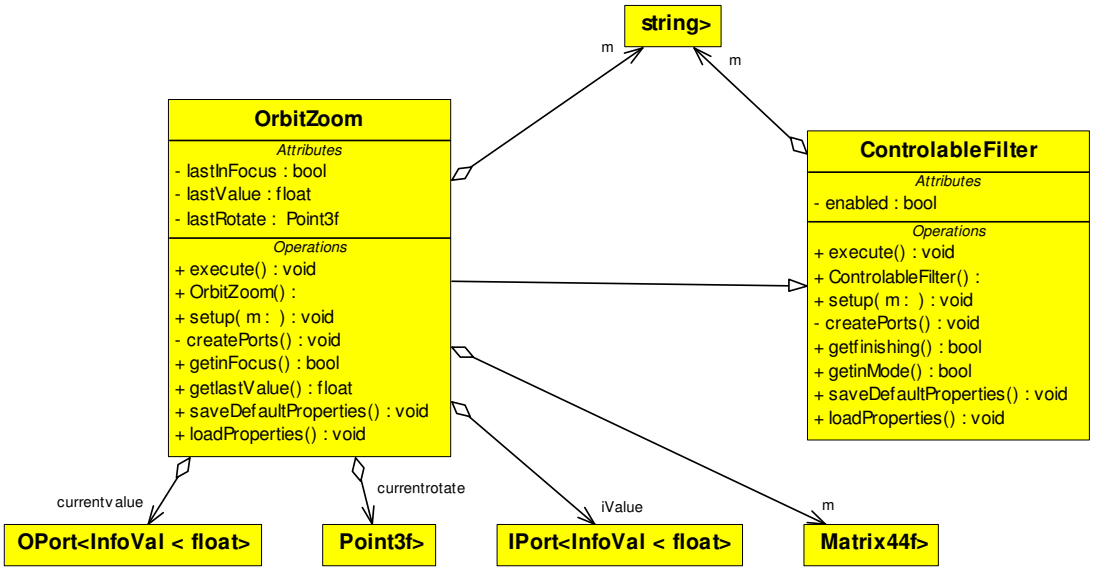


**Imagen 32 Diagrama de Clases Filtro OrbitZoom**

Al igual que en los anteriores, el se crean los puertos en el método createPorts(), en este caso se tienen tres puertos de entrada y dos de salida (Tabla 14).

**Tabla 14 Método createPorts de OrbitZoom (OrbitZoom.cxx Línea 68)**

```
void OrbitZoom::createPorts()
{
    onoff = new IPort<InfoVal<bool> >( this, "onoff" );
    iports.add( onoff );
    m = new IPort<Info< gmtl::Matrix44f> >( this, "m" );
    iports.add( m );
    iValue =  new IPort<InfoVal<float> >( this, "iValue" );
    iports.add( iValue );
    currentvalue =  new OPort<InfoVal<float> >( this, "currentvalue" );
    oports.add( currentvalue );
    currentrotate =  new OPort<Info<gmtl::Point3f> >( this, "currentrotate" );
    oports.add( currentrotate );
}
```

La ejecución de este filtro está basada en el valor de la matriz que llega al puerto 'm', obteniendo los valores de rotación por medio de las funciones de gmtl: EulerAngleXYZf y makeRot (Tabla 15).

**Tabla 15 Metodo execute de OrbitZoom (OrbitZoom.cxx Línea 24)**

```
void OrbitZoom::execute()
{
   if (onoff->hasInfo()){
     InfoVal<bool> *info = onoff->popTyped();
     if (!( info->getInfo() ) ){
          return;
     }
   }
   ControlableFilter::execute();
   if( iValue->hasInfo() )
   {
     InfoVal<float> *info = iValue->popTyped();
     currentvalue->push( info );
   }
   if( m->hasInfo() )
   {
```

```
    Info<gmtl::Matrix44f>* info = m->popTyped();
    gmtl::Point3f p;
    p(0) = (gmtl::makeRot<gmtl::EulerAngleXYZf>( *info->getInfo() ) )(0);
    p(1) = (gmtl::makeRot<gmtl::EulerAngleXYZf>( *info->getInfo() ) )(1);
    p(2) = (gmtl::makeRot<gmtl::EulerAngleXYZf>( *info->getInfo() ) )(2);
    gmtl::Point3f* p2 = new gmtl::Point3f( p );
    Info<gmtl::Point3f>* info2 = new Info<gmtl::Point3f>( p2 );
    info2->addRef();
    currentrotate->push( info2 );
    info->delRef();
   }
}
```

- PerformerFile: Carga un archivo de un modelo Performer y crea el nodo correspondiente de translación y rotación, adicionalmente ofrece cambiar estos valores por medio de setrotate y setzoom.

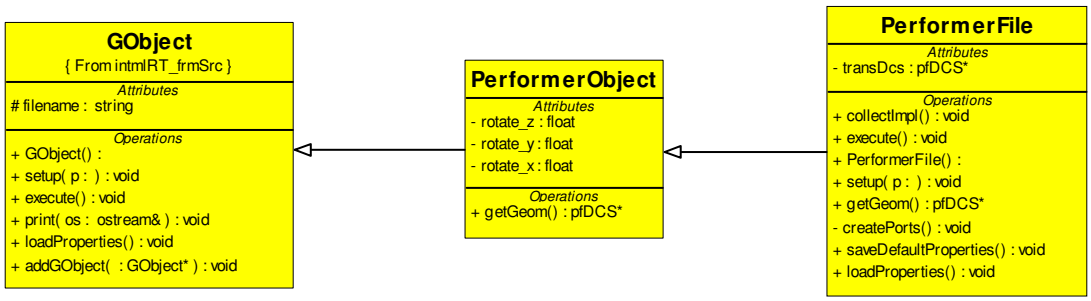


**Imagen 33 Diagrama de Clases Filtro PerformerFile**

El filtro PerfomerFile tiene como objetivo representar un objeto Performer, por lo cual hereda de PerformerObject[12] (Imagen 33), y la forma de hacerlo es cargando el objeto desde un archivo en el sistema.

**Tabla 16 Método createPorts de PerformerFile (PerformerFile.cxx Línea 70)**

```
void PerformerFile::createPorts()
{
  // Add IPort of type float
  setzoom = new IPort<InfoVal<float> >( this, "setzoom" );
  iports.add( setzoom );
  // Add IPort of type float
```

[12] PerformerObject es una clase Abstracta

```
  setrotate = new IPort<Info<gmtl::Point3f > >( this, "setrotate" );
  iports.add( setrotate );
}
```

El método createPorts (Tabla 16) inicializa los puertos de entrada setZoom y setRotate los cuales permiten realizar las transformaciones en el objeto como se ve en la siguiente tabla (Tabla 17).

**Tabla 17 Metodo execute de PerformerFile (PerformerFile.cxx Línea 48)**

```
void PerformerFile::execute()
{
  GObject::execute();
  if( setzoom->hasInfo() )
    {
      InfoVal<float> *info = setzoom->popTyped();
      float temp = info->getInfo();
      info->delRef();
      transDcs->setTrans(0.0, -temp, 0.0);
    }
  if( setrotate->hasInfo() )
    {
      Info<gmtl::Point3f> *info = setrotate->popTyped();
      gmtl::Point3f position = *info->getInfo();
      info->delRef();
      rotate_x = position(0)*360;
      rotate_y = position(1)*360;
      rotate_z = position(2)*360;
      transDcs->setRot(rotate_x,rotate_y,rotate_z);
    }
}
```

- SaveFile: Cuando este filtro se encuentra activo guarda el estado de las variables del usuario en un archivo, para poder recuperar posteriormente el mismo estado.

La aplicación une estos filtros, y obtiene el visualizador, donde se interactúa con un tracker, el primero, para zoom - rotate y el segundo para salvar el estado. A continuación se mostrará el código que representa la aplicación anterior y una descripción de cada una de sus secciones.

**Tabla 18 Código de la Aplicación Generic Performer Viewer (GenericPerformer.cxx Línea 48)**

```
1 void GenericPerformer::load()
2 {

3   Device* tr1 = new FobTracker( );
4   tr1->loadProperties( );
5   std::map<std::string,std::string> track1;
6   track1("name") = "LeftHand";
7   tr1->setup( track1 );
8   idevices.push_back( tr1 );

9   Device* tr2 = new FobTracker( );
10  tr2->loadProperties( );
11  std::map<std::string,std::string> track2;
12  track2("name") = "RightHand";
13  tr2->setup( track2 );
14  idevices.push_back( tr2 );

15  GObject* perf1 = new PerformerFile( );
16  perf1->loadProperties( );
17  std::map<std::string,std::string> per1;
18  per1("name") = "PerformerFile";
19  per1("filename")="/usr/share/Performer/data/tux_sit.pfb";
20  perf1->setup( per1 );
21  objects.push_back( perf1 );

22  std::map<std::string,std::string> z1;
23  z1("name") = "zone1";
24  InT* zone1 = new IsInZone( );
25  zone1->loadProperties( );
26  zone1->setup( z1 );
27  ints.push_back( zone1 );

28  std::map<std::string,std::string> z2;
29  z2("name") = "zone2";
30  InT* zone2 = new IsInZone( );
31  zone2->loadProperties( );
32  zone2->setup( z2 );
33  ints.push_back( zone2 );

34  std::map<std::string,std::string> orb1;
35  orb1("name") = "Orbit";
36  InT* Orbit = new OrbitZoom( );
37  Orbit->loadProperties( );
38  Orbit->setup( orb1 );
39  ints.push_back( Orbit );
```

```
40  gmtl::Point3f *minZone1=new gmtl::Point3f( 1, -3, 0);
41  gmtl::Point3f *maxZone1=new gmtl::Point3f( 4.5, -1, 4.7);
42  Push(new Info<gmtl::Point3f>(minZone1),zone1,"pointmin");
43  Push(new Info<gmtl::Point3f>(maxZone1),zone1,"pointmax");

44  gmtl::Point3f *minZone2 = new gmtl::Point3f( 1, 1, 0);
45  gmtl::Point3f *maxZone2 = new gmtl::Point3f( 4, 3, 4.7);
46  Push(new Info<gmtl::Point3f>(minZone2),zone2,"pointmin");
47  Push(new Info<gmtl::Point3f>(maxZone2),zone2,"pointmax");
48  Connect( tr1, "matrix", zone1, "m" );
49  Connect( tr2, "matrix", zone2, "m" );
50  Connect( zone1, "value", Orbit, "iValue");
51  Connect( zone1, "inFocus", Orbit, "onoff");
52  Connect( Orbit, "currentvalue", perf1 , "setzoom");
53  Connect( Orbit, "currentrotate", perf1 , "setrotate");
```

- Líneas 3-8: creación del filtro para el tracker#1
- Líneas 9-14: creación del filtro para el tracker#2
- Líneas 15-21: Creación del objeto Performer, en este caso está cargando el objeto tux_sit.pfb, el cual viene como ejemplo en la distribución libre de Performer
- Líneas 22-27: Creación del filtro IzInZone1
- Líneas 28-33: Creación del filtro IzInZone2
- Líneas 34-39: Creación del filtro Orbit
- Líneas 40-43: Creación de los límites máximos y mínimos del filtro IsinZone1 y conexión al filtro
- Líneas 44-47: Creación de los límites máximos y mínimos del filtro IsinZone2 y conexión al filtro
- Línea 48: Conexión del tracker1 con el puerto IzInZone1
- Línea 49: Conexión del tracker2 con el puerto IzInZone2
- Línea 50-51: Conexión del filtro IzInZone1 con el Orbit IzInZone1
- Línea 52-53: Conexión del Orbit con el objeto Performer

## 4.2. Visualizador Performer con Posición de Cabeza

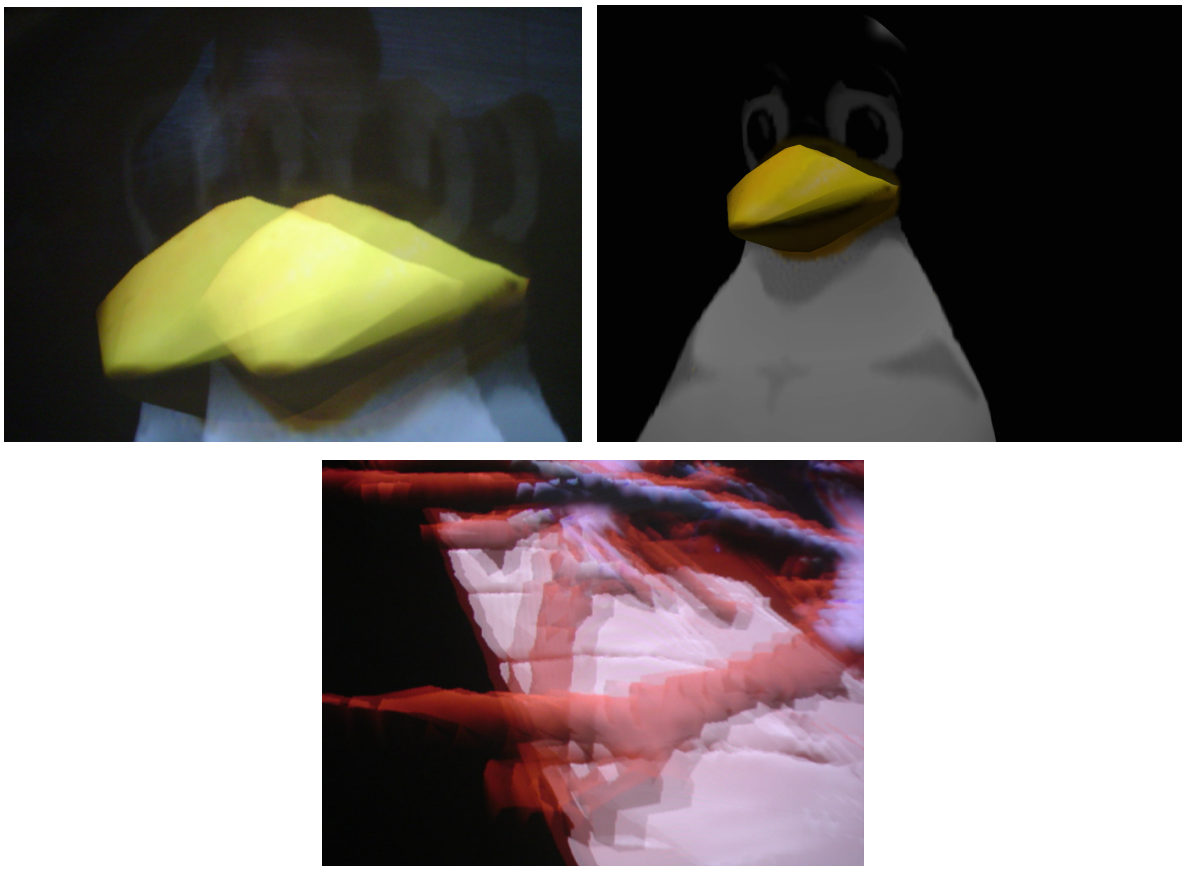

**Imagen 34 Imágenes de las Aplicaciones en Ambiente Fish Tank Desktop**

Para modificar la aplicación anterior y permitir que se visualice en un ambiente Fish Tank Desktop es necesario incluir el punto de vista de usuario, esto se logra ubicando otra conexión entre un nuevo dispositivo, en este caso otro tracker y el punto de vista de la aplicación, como se puede observar en la imagen siguiente (Imagen 35).

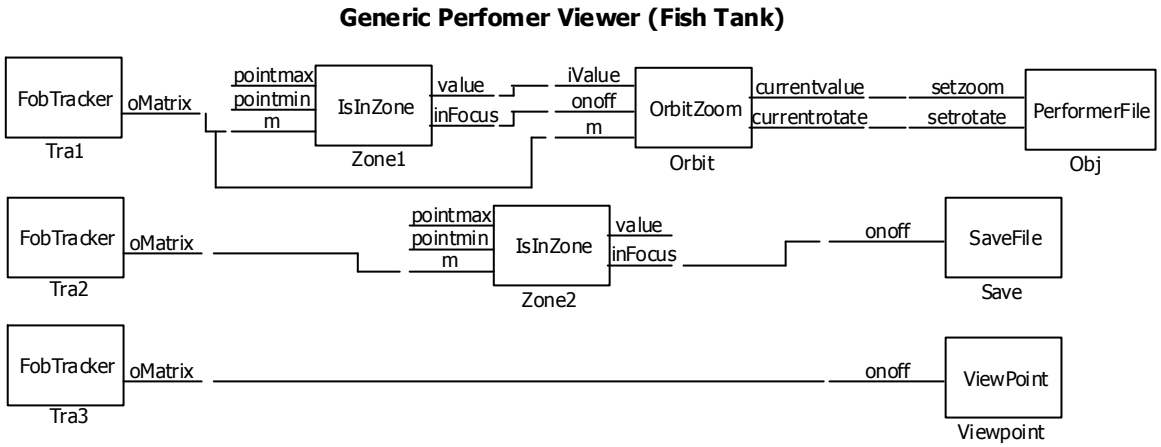


**Imagen 35 Visualizador de Objetos Performer en un Fish Tank**

En la actualidad VrJuggler no soporta la re-configuración dinámica de sus archivos, por lo tanto la forma de crear este ambiente fue por medio de la modificación en los archivos de configuración de VrJuggler. Estos cambios se deben realizar los siguientes pasos.

- Agregar un nuevo tracker al sistema
- Agregar un nuevo proxy de posición relacionando en nuevo tracker
- Agregar un nuevo Alias para el proxy
- En la sección de visualización agregar un nuevo plugin de simulación y relacionar el alias con la posición de la cámara (cameraPos).

El archivo XML debe quedar de la siguiente manera:

```
<simPlugIn>
 <default_simulator name="simPlugIn">
  <cameraPos>SimCam1 Proxy</cameraPos>
  <drawProjections>true</drawProjections>
  <surfaceColor>0.5</surfaceColor>
  <surfaceColor>0.3</surfaceColor>
  <surfaceColor>0.4</surfaceColor>
 </default_simulator>
</simPlugIn>
```

## 4.3. Visualizador DCom

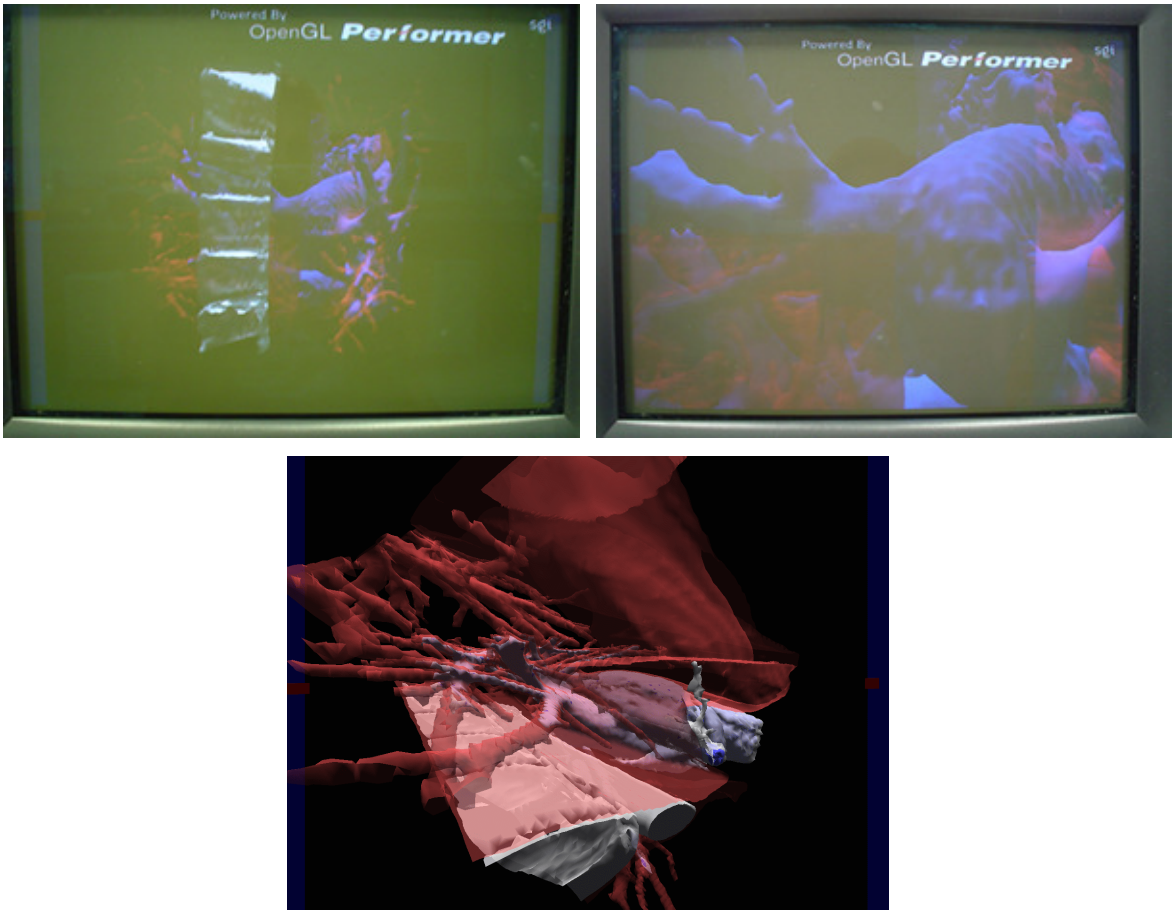


**Imagen 36 Imágenes de la Aplicación Visualizador DCom**

Esta aplicación es menos genérica que la anterior, pero permite más opciones, ya que no sólo es un visualizador sino que permite modificar los parámetros de reconstrucción, esta aplicación utiliza muchos de los filtros de la anterior aplicación y otros nuevos que se especificarán a continuación.

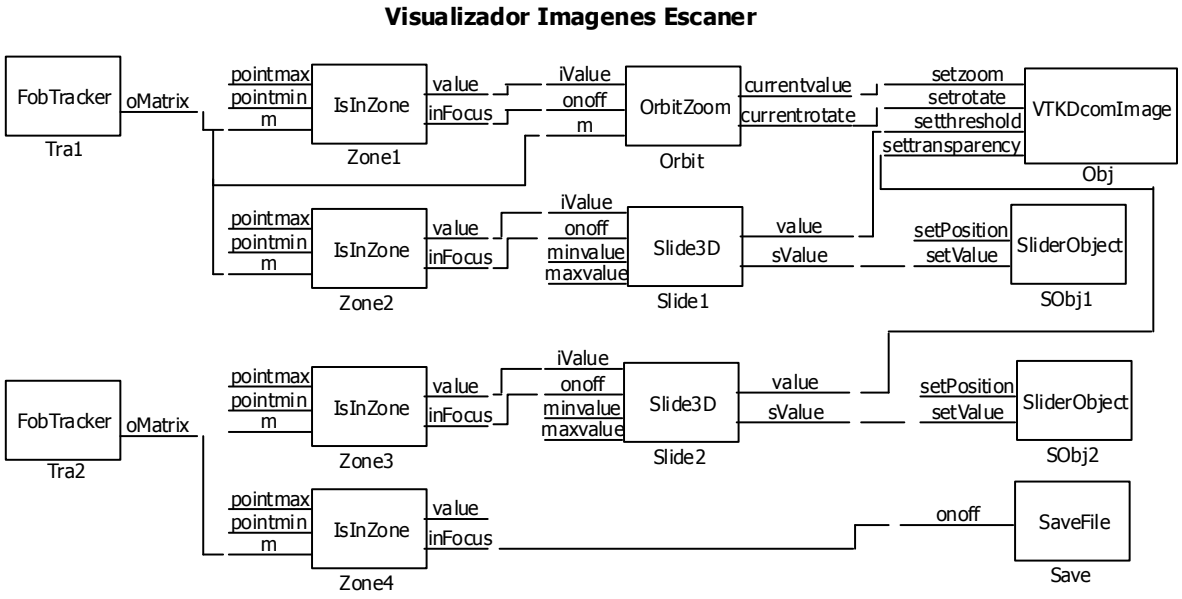


**Imagen 37 Visualizador de Imágenes DCOM**

- Slide3D: Se encarga de la lógica de un slider, calculando su posición y enviándola de dos maneras, la primera constantemente y la segunda al salir del slider (value, sValue).

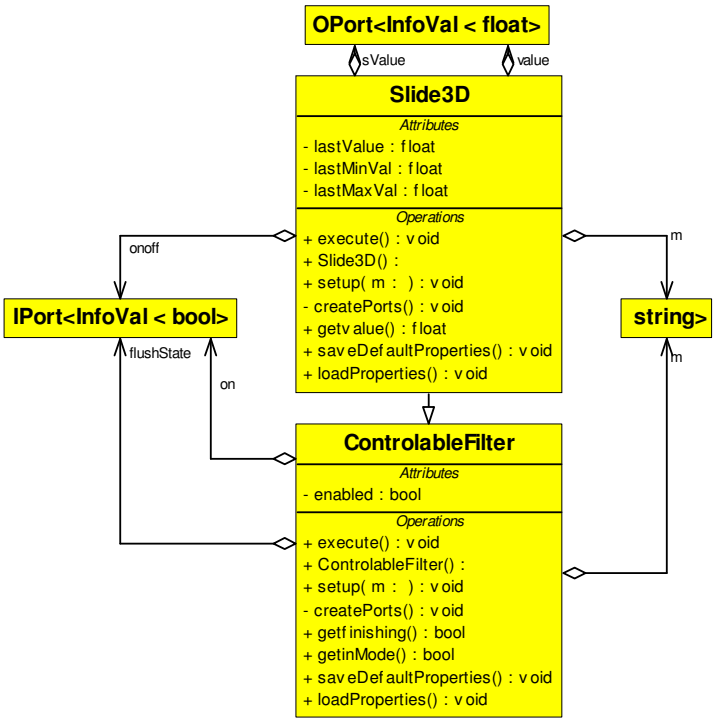


**Imagen 38 Diagrama de Clases Filtro Slide3D**

- SliderObject: Es la forma de un slider generado en VTK (vtkCubeSource), y transformado a Performer por vtkActorToPf.

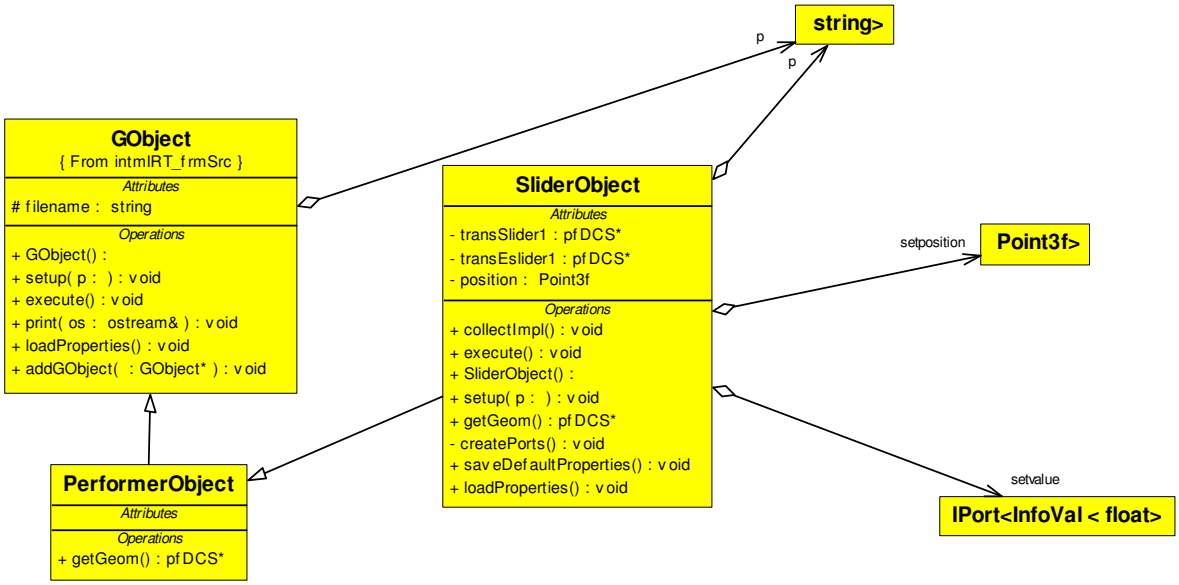


**Imagen 39 Diagrama de Clases Filtro SlideObject**

- VTKDComImage: Este filtro reconstruye dos superficies de acuerdo a dos umbrales establecidos, una de estas superficies es fija y la otra con posibilidad de cambiar su umbral de reconstrucción y su transparencia.

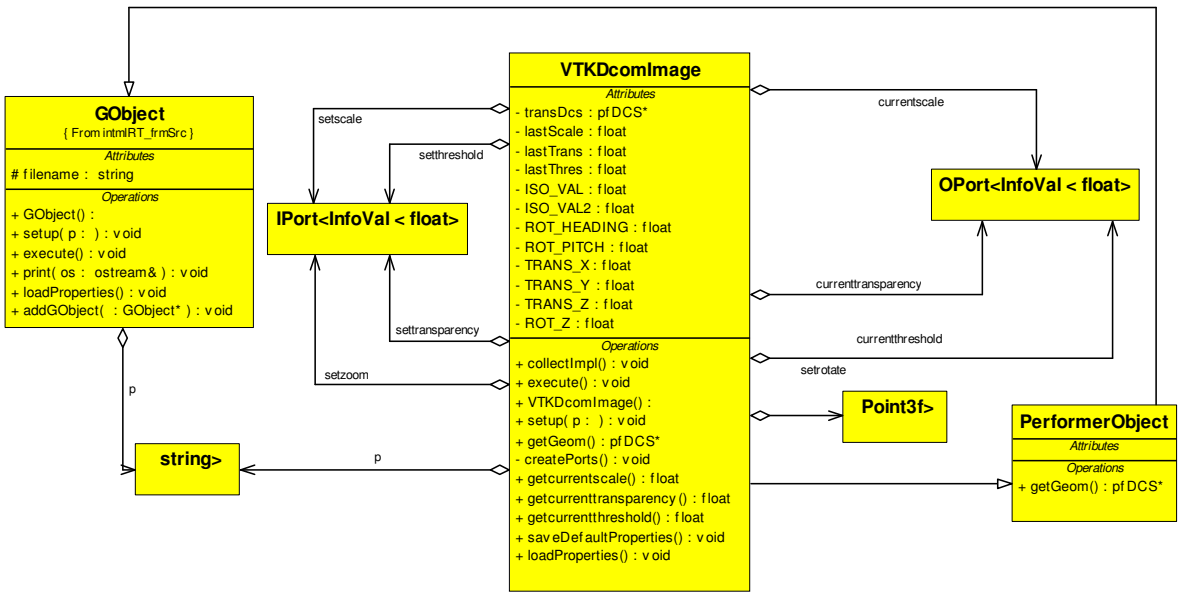


**Imagen 40 Diagrama de Clases Filtro VTKDComImage**

Toda la reconstrucción se hace por medio de VTK y su respectiva transformación en vtkActorToPf.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE App PUBLIC "-//pfiguero//3D Interaction Techniques ML//EN"
        "http://localhost/~pfiguero/InTmlTemp/spec/InTml.dtd">
3  <App id="imageApp.thresholdDCom">
4   <ShortDesc>Dicom Application</ShortDesc>
5   <Import id="imageApp"/>
6   <Import id="imageTest"/>
7   <!-- Devices -->
8   <IDevice id="keyboard" type="GenericKeyboard"/>
9   <IDevice id="tracker1" type="FobTracker"/>
10      <IDevice id="tracker2" type="FobTracker"/>
11      <IDevice id="tracker3" type="FobTracker"/>
12  <!-- Devices -->
```

```
13      <ObjectHolder id="theCurrentViewpoint"/>
14      <Binding iE="tracker3" iP="q" oE="theCurrentViewpoint" oP="setQ"/>
15 <Binding iE="tracker3" iP="p" oE="theCurrentViewpoint" oP="setPos"/>
16 <!-- Scene definition -->
17 <Object id="medicalScene" type="Scene"/>
18 <!-- VTKDcomImage object for dicom files -->
19 <Object id="data" filename="medical/image23" type="VTKDcomImage"/>
20 <!-- Binding to scene -->
21 <Binding iE="_self" iP="data" oE="medicalScene" oP="addObject"/>
22 <!-- Constants -->
23 <Constant id="pV" type="Pos3D" value="0 0 0"/>
24 <Constant id="qV" type="Quaternion" value="0 0 -1 0"/>
25 <Constant id="notVisible" type="boolean" value="false"/>
26 <!-- Viewpoint and bindings -->
27 <Object id="viewpoint" filename="" type="Viewpoint"/>
28 <Binding iE="_self" iP="pV" oE="viewpoint" oP="setPos"/>
29 <Binding iE="_self" iP="qV" oE="viewpoint" oP="setQ"/>
30 <Binding iE="_self" iP="notVisible" oE="viewpoint" oP="setVisible"/>
40 <!-- Link it to the viewpoint in the system -->
41 <Binding iE="_self" iP="viewpoint" oE="theCurrentViewpoint"
        oP="object"/>
42 <!-- Transparency Zone (test values for zones) -->
43 <Filter id="inZone" type="IsInZone"/>
44 <Constant id="pointMin1" type="Pos3D" value="2 5 8"/>
45 <Constant id="pointMax1" type="Pos3D" value="4 5 2"/>
46 <Binding iE="tracker1" iP="p" oE="inZone" oP="p"/>
47 <Binding iE="tracker1" iP="q" oE="inZone" oP="q"/>
48 <Binding iE="_self" iP="pointMin1" oE="inZone" oP="pointmin"/>
49 <Binding iE="_self" iP="pointMax1" oE="inZone" oP="pointmax"/>
50 <!-- Slide for transparency -->
51 <Filter id="slide" type="Slide3D"/>
52 <Object id="geometry_ext" filename="geometry/ext_slide.txt"
        type="Geometry"/>
53 <Object id="geometry_int" filename="geometry/int_slide.txt"
54      type="Geometry"/>
55 <Binding iE="tracker1" iP="p" oE="slide" oP="p"/>
56 <Binding iE="tracker1" iP="q" oE="slide" oP="q"/>
57 <Binding iE="_self" iP="geometry_ext"
58       oE="slide" oP="slideGeomInt"/>
59 <Binding iE="_self" iP="geometry_int"
60       oE="slide" oP="slideGeomExt"/>
61 <Binding iE="inZone" iP="minval" oE="slide" oP="minval"/>
62 <Binding iE="inZone" iP="maxval" oE="slide" oP="maxval"/>
63 <Binding iE="inZone" iP="inFocus" oE="slide" oP="onoff"/>
64 <!-- Transparency slide to VTKDcomImage  object -->
65 <Binding iE="slide" iP="value" oE="data" oP="settransparency"/>
66 <!-- Threshold zone  -->
```

```
67  <Filter id="inZone2" type="IsInZone"/>
68  <Constant id="pointMin2" type="Pos3D" value="2 5 8"/>
69  <Constant id="pointMax2" type="Pos3D" value="4 5 2"/>
70  <Binding iE="tracker2" iP="p" oE="inZone2" oP="p"/>
71  <Binding iE="tracker2" iP="q" oE="inZone2" oP="q"/>
72  <Binding iE="_self" iP="pointMin2" oE="inZone2" oP="pointmin"/>
73  <Binding iE="_self" iP="pointMax2" oE="inZone2" oP="pointmax"/>
74  <!-- Slide for threshold -->
75  <Filter id="slide2" type="Slide3D"/>
76  <Object id="geometry_ext2" filename="geometry/ext_slide2.txt"
        type="Geometry"/>
77  <Object id="geometry_int2" filename="geometry/int_slide2.txt"
        type="Geometry"/>
78  <Binding iE="tracker2" iP="p" oE="slide2" oP="p"/>
79  <Binding iE="tracker2" iP="q" oE="slide2" oP="q"/>
80  <Binding iE="_self" iP="geometry_ext2"
         oE="slide2" oP="slideGeomInt"/>
81  <Binding iE="_self" iP="geometry_int2"
         oE="slide2" oP="slideGeomExt"/>
82  <Binding iE="inZone2" iP="minval" oE="slide2" oP="minval"/>
83  <Binding iE="inZone2" iP="minval" oE="slide2" oP="minval"/>
84  <Binding iE="inZone2" iP="inFocus" oE="slide2" oP="onoff"/>
85  <!-- Threshold slide to VTKDcomImage  object -->
86  <Binding iE="slide2" iP="value" oE="data" oP="setthreshold"/>
87  <!-- Zoom zone -->
88  <Filter id="inZone3" type="IsInZone"/>
89  <Constant id="pointMin3" type="Pos3D" value="2 5 8"/>
90  <Constant id="pointMax3" type="Pos3D" value="4 5 2"/>
91  <Binding iE="tracker2" iP="p" oE="inZone3" oP="p"/>
92  <Binding iE="tracker2" iP="q" oE="inZone3" oP="q"/>
93  <Binding iE="_self" iP="pointMin3" oE="inZone3" oP="pointmin"/>
94  <Binding iE="_self" iP="pointMax3" oE="inZone3" oP="pointmax"/>
95  <!-- Slide for zoom -->
96  <Filter id="orbit" type="OrbitZoom"/>
97  <Binding iE="tracker2" iP="p" oE="orbit" oP="p"/>
98  <Binding iE="tracker2" iP="q" oE="orbit" oP="q"/>
99  <Binding iE="inZone3" iP="inFocus" oE="orbit" oP="on"/>
100     <!-- Orbit behavior to VTKDcomImage object -->
101     <Binding iE="orbit" iP="currentscale" oE="data" oP="setscale"/>
102     <Binding iE="orbit" iP="currentposition" oE="viewpoint" oP="setPos"/>
103     <Binding iE="orbit" iP="currentrotate" oE="viewpoint" oP="setQ"/>
104     <!-- Save zone -->
105     <Filter id="inZone4" type="IsInZone"/>
106     <Constant id="pointMin4" type="Pos3D" value="2 5 8"/>
107     <Constant id="pointMax4" type="Pos3D" value="4 5 2"/>
108     <Binding iE="tracker2" iP="p" oE="inZone4" oP="p"/>
109     <Binding iE="tracker2" iP="q" oE="inZone4" oP="q"/>
```

```
110     <Binding iE="_self" iP="pointMin4" oE="inZone4" oP="pointmin"/>
111     <Binding iE="_self" iP="pointMax4" oE="inZone4" oP="pointmax"/>
112     <!-- Save behavior -->
113     <Filter id="save" type="Save"/>
114     <Binding iE="viewpoint" iP="posChanged" oE="save" oP="p"/>
115     <Binding iE="viewpoint" iP="qChanged" oE="save" oP="q"/>
116     <Binding iE="inZone4" iP="inFocus" oE="save" oP="onoff"/>
117     <Binding iE="data" iP="currenttransparency"
          oE="save" oP="settransparency"/>
118     <Binding iE="data" iP="currentscale" oE="save" oP="setscale"/>
119     <Binding iE="data" iP="currentthreshold"
          oE="save" oP="setthreshold"/>
120</App>
```

- Líneas 9-11: Creación de los filtros que representan los trackers
- Línea 17: Definición de la escena
- Línea 19-21: Creación del objeto VTKDcomImage y unión de este con la escena
- Líneas 23-25, 43-45: Declaración de constantes
- Línea 43: Creación del primer filtro IzInZone
- Líneas 44-47: Inicialización del filtro uniendo las constantes por medio del conector
- Líneas 50-54: Creación del primer Slide3D y la correspondiente inicialización de sus valores
- Líneas 67-69: Creación de la segunda zona de interacción
- Líneas 78-86: Conexiones de los filtros.
- Línea 88: Creación de la tercera zona
- Línea 96: Creación del OrbitZoom
- Línea 105: Creación de la última zona
- Línea 113: Filtro para salvar
- Líneas 114-117: conexiones para modificar el objeto.

# 5. CONCLUSIONES Y TRABAJO FUTURO

Durante el desarrollo de esta investigación se lograron analizar las diferentes herramientas de software que existen en la actualidad para el desarrollo de aplicaciones de RV, estableciendo sus fortalezas y debilidades. Esto permitió que se pudiera establecer un conjunto de herramientas que se complementaran entre si, y así satisfacer las necesidades planteadas para la implementación del ambiente de trabajo propuesto.

Adicionalmente, el uso de InTml como la arquitectura de nuestras aplicaciones nos permite vislumbrar posibilidades de reutilización de código y de desarrollo acelerado de aplicaciones, basado en librerías de filtros.

Como consecuencia de lo anterior, es decir, la integración del conjunto de herramientas de software seleccionadas y la aplicación de la arquitectura, los resultados obtenidos cumplen con las expectativas de un ambiente propicio para el desarrollo de aplicaciones de RV.

Una vez generada la infraestructura los ejemplos que se construyeron permitieron la validación de la misma, ya que se lograron implementar cambios a partir del ejemplo original sin traumatismos en el desarrollo, esto se debió a la reutilización y adaptación de los filtros de acuerdo a la evolución de los requerimientos de los diferentes ejemplos.

Los primeros ejemplos desarrollados en el ambiente obedecían a la necesidad de validación de la propuesta, sin embargo, una vez se estableció que el ambiente cumplía con las expectativas, se dio paso a la implementación de una aplicación con requerimientos reales. Esta aplicación fue la reconstrucción de modelos por medio de imágenes médicas, obtenidas a través de scanner, lo cual permite visualizar el potencial del ambiente propuesto al apoyar diferentes procesos en diversas áreas de investigación.

Comprobada la utilidad del ambiente propuesto se establecieron procedimientos claros de instalación de los componentes del sistema, con el objeto de facilitar la replicación del ambiente a futuro. Esta visión de futuro es razonable dado que se comprobó la conveniencia y el potencial de la infraestructura, y se logró con costos muy inferiores a los de ambientes comerciales como los descritos en este documento. Esto hace que esta infraestructura esté más acorde con la realidad colombiana y por ende sea mas asequible y viable su adquisición.

Adicionalmente a los resultados descritos, esta investigación permitió adaptar el framework InTml para el ambiente planteado; ya que solo se había probado la versión para java 3D implementada por Pablo Figueroa en la Universidad de Alberta – Canadá. Así mismo, los ejemplos generados contribuyen a la compresión de la metodología y a su evolución, al mismo tiempo que complementan el desarrollo de la investigación sobre el área de RV.

Por último, esta investigación plantea diversos frentes de ampliación de conocimiento en el área, específicamente en la evolución de la plataforma como tal. Algunos de estos frentes se enuncian a continuación:

Uno de los principales trabajos a llevar a cabo que no se encontraba dentro de los alcance de la tesis, es la validación del cambio de dispositivos, por lo tanto sería muy útil implementar nuevos filtros que representen otros dispositivos y probar diferentes configuraciones entre estos. Lo anterior, ampliará el banco de filtros desarrollados lo que permitirá maximizar la reutilización y aplicabilidad del ambiente, ya que se podrán desarrollar aplicaciones mas complejas al contar con una gran diversidad de componentes que representan los diferentes elementos del sistema.

Otros trabajos que complementarían está investigación son: la construcción de una herramienta de generación de código para la infraestructura propuesta similar a la herramienta existente para Java3D, la cual produce el código fuente a partir de los archivos XML que describen la aplicación; y el desarrollo de una interfaz gráfica que permita la

elaboración de aplicaciones en el framework InTml, construido como parte de la propuesta de esta investigación, de una manera mas amigable para el usuario final.

Para finalizar, sería provechoso migrar las aplicaciones existentes en el framework de java 3D al framework propuesto validando la portabilidad del ambiente definido.

# 6. BIBLIOGRAFÍA

**[ALI 04]** Carnegie Mellon University and University of Virginia. Alice: Easy interactive 3D graphics. http://www.alice.org, 1999.

**[ALL 02]** Jeremie Allard, Valerie Gouranton, Loick Lecointre, Emmanuel Melin, Bruno Raffin; Net Juggler and SoftGenLock: Running VR Juggler with Active Stereo and Multiple Displays on a Commodity Component Cluster; Proceedings of the IEEE Virtual Reality; 2002.

**[ARS 03]** Ars Electronica Futurelab http://futurelab.aec.at/arsbox/index2.htm; ArsBox, consultado Agosto 2003.

**[BAR 03]** Bruno Barberi Gnecco DiceLib DIstributed CAVE Engine; http://www.lsi.usp.br/~brunobg/dicelib/; Consultada Septiembre de 2003.

**[BAR 04]** Barco; http://www.barco.com/; Consultada Julio 2004.

**[BRY 96]** Steve Bryston; Virtual Reality in Scientific Visualization; Communications of the ACM; 1996.

**[CYB 03]** The Cybersphere; http://www.vr-systems.ndtilda.co.uk/sphere1.htm; Consultada Agosto 2003.

**[DES 04]** Embarcadero Technologies, Describe; http://www.embarcadero.com/products/describe/dedatasheet.html, Consultado: Julio de 2004

**[FAK 04]** FakeSpace Systems; http://www.fakespacesystems.com/index.htm; Consultada en Junio de 2004;

**[FIG 02]** Pablo Figueroa, Mark Green, H. James Hoover; InTml: A Description Language for VR Applications; COMMUNICATIONS OF THE ACM; pag 53-58; 2002

**[FIG 04]** Pablo Figueroa. Retargeting of Virtual Reality Applications. PhD thesis, University of Alberta, 2004.

**[FOR 97]** Nigel Foreman, Paul Wilson,Danae Stanton; Vr and Spatial Awareness in disabled children; COMMUNICATIONS OF THE ACM; pag 76-77; 1997.

**[FOR 01]** Andrew S. Forsberg, David H. Laidlaw, Robert M. Kirby, George E. Karniadakis, Jonathan L. Elion; Immersive Virtual Reality for Visualizing Flow Through an Artery; 457-461.

**[GRE 98]** Walter J. Greenleaf; Applying Vr to physical medicine and rehabilitation; COMMUNICATIONS OF THE ACM; pag 42-46; 1998.

**[GRO 98]** Markus H. Gross; Computer Graphics in Medicine: From Visualization to Surgery Simulation; Computer Graphics; Pag 53-57; 1998.

**[GRO 03]** Markus Gross, Stephan Würmlin, Martin Naef, Edouard Lamboray, Christian Spagno, Andreas Kunz, Esther Koller-Meier, Tomas Svoboda, Luc Van Gool, Silke Lang, Kai Strehlke, Andrew Vande Moere, Oliver Staadt; blue-c: A Spatially Immersive Display and 3D Video Portal for Telepresence; 819-827; 2003;

**[JAC 99]** Robert Jacob, Leonidas Deligiannidis, Stephen Morrison; A Software Model and Specification Language for Non-WIMP User Interfaces; ACM Transactions on Computer-Human Interaction; 1999; Pag 1-46.

**[JAC 02]** Jeffrey Jacobson and Zimmy Hwang; Unreal Tournament for Immersive Interactive Theater; Communications of the ACM; 2002.

**[JAC 03]** Cave Unreal Tournament; http://planetjeff.net/ut/CaveUT.html Consultada Agosto de 2003.

**[JUS 02]** Christopher Just; 2002; Cluster Juggler; Virtual Reality Applications Center.

**[LEW 02]** Michael Lewis, Jeffrey Jacobson; Game Engines in Scientific Research; Communications of the ACM; 2002.

**[LIU 01]** Damon Liu, Mark Burgin, Walter Karplus, Daniel Valentino; Large-Scale Flow Field Visualization for Aneurysm Treatment;
pag 68-72; 2001.

**[MUL 04]** MultiGen Paradigm; http://www.multigen.com/index.shtml; Consultada Junio de 2004.

**[NEI 01]** Carolina Cruz-Neira, Daniel J. Sadin, Tomas A.DeFanti, Robert V Kenyon, John C. Hart; Audio Visual Experience Automatic Virtual Environment; COMMUNICATIONS OF THE ACM; pag 64-72, 1997.

**[NCS 03]** NCSA Visualization and Virtual Environments Projects; http://www.ncsa.uiuc.edu/Divisions/CSD/VisVE/; consultada septiembre 2003.

**[PAP 03]** The CAVE Virtual Reality System; http://www.evl.uic.edu/pape/CAVE/; Consultada Agosto 2003.

**[PER 03]** Joaquín Pérez Martín; Infografía y multimedia; Historia; http://www2.canalaudiovisual.com/ezine/books/contjjpm/intreavirtu.htm; Consultado Octubre de 2003.

**[PER 04]** Lynellen D.S. Perry, Christopher M. Smith, Steven Yang; Una Investigación sobre Interfaces Actuales de Realidad Virtual; http://www.acm.org/crossroads/espanol/xrds3-3/vrhci.html; Consultada junio 2004

**[PON 03]** Michal Ponder,Bruno Herbelin, Tom Molet, Sebastien Schertenlieb, Branislav Ulicny, George Papagiannakis, Nadia Magnenat-Thalmann, Daniel Thalmann}; Immersive VR Decision Training: Telling Interactive Stories Featuring Advanced Virtual Human Simulation Technologies; The Eurographics Association; pag97-106; 2003.

**[POS 96]** Timothy Poston, Luis Serra; Dextrous (Virtual Work): A system for visualizing and manipulating medical images is detailed, with emphasis on interaction techniques; COMMUNICATIONS OF THE ACM; pag 37-45; 1996.

**[RAJ 03]** Paul Rajlich; Cave Quake; National Center for Supercomputing Applications; http://brighton.ncsa.uiuc.edu/~prajlich/caveQuake/; consultada Septiembre de 2003.

**[RAJ 04]** Paul Rajlich; vtkActorToPF; National Center for Supercomputing Applications; http://brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF; consultada 2004.

**[SAR 03]** Sara "Computing & Networking Service"; Virtual Reality Facilities http://www.sara.nl/products/products_08a_eng.html; Consultado Agosto 2003.

**[SCH 01]** Benjamin Schaeffer; A Software System for Inexpensive VR via Graphics Clusters; 2001.

**[SCH 02]** Benjamin Schaeffer; Networking and Management Frameworks for Cluster-Based Graphics; pag 1-8; 2002.

**[SCH 03]** Benjamin Schaeffer, Camille Goudeseune; Syzygy: Native PC Cluster VR;2003.

**[SGI 04]** OpenGL Performer SGI http://www.sgi.com/software/performer/ Consultada Enero de 2004

**[STE 03]** Ch. Stern, H. Noser, J. Weissmann1, P. Stucki; Application Scenarios for Scientific Visualization and Virtual Reality Using a CAVE Infrastructure; The Eurographics Association; pag, 319-320; 2003.

**[SYS 03]** SYSYGY A Toolkit for Virtual Reality on PC Clusters; http://www.isl.uiuc.edu/syzygy.htm; Consultado Septiembre 2003.

**[TAN 01]** Vildan Tanriverdi, Robert J.K. Jacob; VRID: A Design Model and Methodology for Developing Virtual Reality Interfaces; COMMUNICATIONS OF THE ACM; pag 175-182; 2001.

**[VAL 01]** Raquel Valdés C., Oscar Yáñez S; Superficies Activas con Splines Cúbicos para Segmentación Semi-automática de Tráquea; Memorias II Congreso Latinoamericano de Ingeniería Biomédica; 2001.

**[VIC 04]** VICO; Patrones de diseño (notación UML) http://www.vico.org/pages/PatronsDisseny.html; Consultada Junio 2004

**[VIS 03]** VisBox INC. "High–end Display and interaction technologies" http://www.visbox.com/; Consultada Septiembre 2003

**[VRC 04]** VRCO CaveLib; http://www.vrco.com/CAVE_USER/index.html; Consultado Septiembre 2004.

**[VRJ 03]** VrJuggler Open Source Virtual Reality http://www.vrjuggler.org Consultada Diciembre de 2003.

**[VRP 04]** VRPN Virtual Reality Peripheral Network;

http://www.cs.unc.edu/Research/vrpn/; Consultado Enero 2004

**[VTK 04]** The Visualization Toolkit VTK; http://vtk.org/; Consultada 2004

**[WEB 04]** Web 3D Consortium X3D; http://www.web3d.org/x3d/overview.html Consultado Enero de 2004

**[WIL 04]** James Willans; Marigold Toolset; http://www-users.cs.york.ac.uk/~jwillans/VE_Research.html; Consultada julio de 2004.

**[ZHA 01]** S. Zhang, M. DaSilva, P.J. Basser, C. Pierpaoliy, E.A. Chiocca; An Immersive Virtual Environment for DT-MRI Volume Visualization Applications: a Case Study; pag 437-440, 584; 2001.

**[ZUF 03]** Marcelo Knörich Zuffo, Hank Kaczmarski, Bruno Raffin; Commodity Clusters for Immersive Projection Environments; SIGGRAPH 2003; 2003.

# APÉNDICES

## Pasos para configurar un ambiente fish tank en Linux

1. Instalación Linux (Fedora Core 1). (fedora.redhat.com)
2. Instalación Drivers Nvidia (1.0-6106). (www.nvidia.com)
3. Configuración del servidor X. (www.xfree86.org)
4. Instalación de VrJuggler (V2 alpha3). (www.vrjuggler.org)
5. Instalación de Performer (V3.1). (www.sgi.com/software/performer/)
6. Instalación de VTK (V4.2). (www.vtk.org)
7. Instalación de VtkActorToPF (V6.8). (brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF).
8. Instalación del framework Intml (). Pablo Figueroa
9. Instalación de Dispositivos (Trackers) en VRJuggler.
10. Instalación para gafas Stereo
11. Configuración de las aplicaciones.
12. Correr la aplicación.

**Archivo de Configuración XFREE86**

```
Section "ServerLayout"
#    Option        "Clone" "on"
     Identifier     "Multihead layout"
     Screen         "Screen0" LeftOf "Screen1"
     Screen         "Screen1" 0 0
     InputDevice    "Mouse0" "CorePointer"
     InputDevice    "Keyboard0" "CoreKeyboard"
     InputDevice    "DevInputMice" "AlwaysCore"
     Option        "Xinerama" "off"
     Option        "Clone" "on"
EndSection

Section "Files"
     RgbPath      "/usr/X11R6/lib/X11/rgb"
     FontPath     "unix/:7100"
EndSection

Section "Module"
     Load  "dbe"
     Load  "extmod"
     Load  "fbdevhw"
     Load  "glx"
     Load  "record"
     Load  "freetype"
     Load  "type1"
EndSection

Section "InputDevice"
     Identifier  "Keyboard0"
     Driver      "keyboard"
     Option        "XkbRules" "xfree86"
     Option        "XkbModel" "pc105"
     Option        "XkbLayout" "la"
EndSection

Section "InputDevice"
     Identifier  "Mouse0"
     Driver      "mouse"
     Option        "Protocol" "IMPS/2"
     Option        "Device" "/dev/psaux"
     Option        "ZAxisMapping" "4 5"
     Option        "Emulate3Buttons" "no"
EndSection
```

```
Section "InputDevice"
     Identifier  "DevInputMice"
     Driver      "mouse"
     Option        "Protocol" "IMPS/2"
     Option        "Device" "/dev/input/mice"
     Option        "ZAxisMapping" "4 5"
     Option        "Emulate3Buttons" "no"
EndSection

Section "Monitor"
     Identifier   "Monitor0"
     VendorName   "Monitor Vendor"
     ModelName    "Dell P1130"
     DisplaySize  400    300
     HorizSync    30.0 - 130.0
     VertRefresh  48.0 - 170.0
     Option        "dpms"
EndSection

Section "Monitor"
     Identifier   "Monitor1"
     VendorName   "Monitor Vendor"
     ModelName    "Dell P1130"
     DisplaySize  400    300
     HorizSync    30.0 - 130.0
     VertRefresh  48.0 - 170.0
     Option        "dpms"
EndSection

Section "Device"
     Identifier  "Nvidia Twin"
     Driver      "nvidia"
     VendorName  "NVidia"
     BoardName   "NVIDIA GeForce4 (generic)"
     Option        "DPMS"
     Option        "Nologo" "on"
     Option        "TwinView" "on"
     Option        "MetaModes" "1280x960, 1280x960"
     Option        "TwinViewOrientation" "LeftOf"
     Option        "ConnectedMonitor" "crt,crt"
     Option        "Stereo" "1"
EndSection

Section "Device"
     Identifier  "Videocard1"
```

```
     Driver      "nv"
     VendorName  "Videocard Vendor"
     BoardName   "NVIDIA Quadro FX (generic)"
EndSection

Section "Screen"
     Identifier "Screen0"
     Device     "Nvidia Twin"
     Monitor    "Monitor0"
     DefaultDepth     24
     SubSection "Display"
          Depth     24
          Modes    "1280x960" "1152x864" "1024x768" "800x600"
"640x480"
     EndSubSection
EndSection

Section "Screen"
     Identifier "Screen1"
     Device     "Nvidia Twin"
     Monitor    "Monitor1"
     DefaultDepth     24
     SubSection "Display"
          Depth     24
          Modes    "1280x960" "1152x864" "1024x768" "800x600"
"640x480"
     EndSubSection
EndSection

Section "DRI"
     Group        0
     Mode         0666
EndSection
```

## Árbol de Directorios CD Anexo

- DistVRJuggler:
  - Makefile
  - App:
    - PerformerFile.o
    - Slide3D.hxx
    - ThresholdDcom.o
    - VRJPFSystem.o
    - IsInZone.hxx
    - OrbitZoom.hxx
    - PerformerObject.cxx
    - Slide3D.o
    - ThresholdDcomOld.cxx
    - VTKDcomImage.cxx
    - ControlableFilter.hxx
    - IsInZone.o
    - PerformerObject.hxx
    - SliderObject.cxx
    - FobTracker.cxx
    - libappIntml.a
    - OrbitZoom.o
    - PerformerObject.o
    - SliderObject.hxx
    - VRJPFApp.cxx
    - VTKDcomImage.hxx
    - FobTracker.hxx
    - Makefile
    - PerformerFile.cxx
    - Properties.cxx
    - SliderObject.o
    - VRJPFApp.hxx
    - FobTracker.o
    - Properties.hxx
    - ThresholdDcom.cxx
    - VRJPFApp.o
    - VTKDcomImage.java
    - Geometry.hxx
    - PerformerFile.hxx
    - Properties.o
    - VRJPFSystem.cxx
    - VTKDcomImage.o
    - IsInZone.cxx
    - OrbitZoom.cxx

    - Slide3D.cxx
    - ThresholdDcom.hxx
    - VRJPFSystem.hxx
- Frm:
    - AbsOPort.o
    - InfoVal.hxx
    - ObjectHolder.cxx
    - StdOut.o
    - AbsPort.cxx
    - InT.cxx
    - ObjectHolder.hxx
    - AbsPort.hxx
    - InT.hxx
    - AbsPort.o
    - Filter.cxx
    - InTmlSystem.cxx
    - OPort.hxx
    - Test1App2.cxx
    - Test1Device.cxx
    - TSApp.cxx
    - AbsApp.cxx
    - Filter.hxx
    - InTmlSystem.hxx
    - PortList.cxx
    - TSApp.hxx
    - AbsApp.hxx
    - ComposedFilter.cxx
    - Filter.o
    - InTmlSystem.o
    - PortList.hxx
    - TSApp.o
    - AbsApp.o
    - ComposedFilter.hxx
    - InT.o
    - PortList.o
    - TSAppTypes.cxx
    - AbsInfo.cxx
    - ComposedFilter.o
    - genTypes.hxx
    - IPort.hxx
    - RefCounter.cxx
    - TSAppTypes.hxx
    - AbsInfo.hxx
    - GObject.cxx
    - libfrmIntml.a
    - RefCounter.hxx

    - TSAppTypes.o
    - AbsInfo.o
    - Device.cxx
    - GObject.hxx
    - Makefile
    - RefCounter.o
    - AbsOPort.cxx
    - Device.hxx
    - GObject.o
    - StdOut.cxx
    - AbsOPort.hxx
    - Device.o
    - Info.hxx
    - StdOut.hxx
- getopts-0.4.1
    - getopts.c
    - getopts.h
    - getopts.o
    - libgetopts.a
    - LICENSE
    - main.c
    - main.o
    - makefile
    - README
- Loader
    - glove.config
    - InTmlLoader.cxx
    - InTmlLoader.o
    - MainInTmlLoader.hxx
    - Makefile
    - sim.base.config
    - sim.wand.mixin.config
    - intmlApp
    - InTmlLoader.hxx
    - MainInTmlLoader.cxx
    - MainInTmlLoader.o
    - sim.base.simple.config
    - tracker.base.simple.config

- Properties:
    - VRJPFSystem.properties

## Arquitectura InTml

Como se comento en el capitulo 3 la arquitectura InTml esta basada en la representación de los objetos de una aplicación como filtros, por lo cual cada filtro tiene una función especifica y particular en el sistema y la aplicación se resume en la comunicación entre todos estos elementos.

Dentro de los filtros es posible definir diferentes elementos como los siguientes:

- Dispositivos de entrada y salida
- Técnicas de interacción
- Animaciones
- Geometría de Objetos

Un ejemplo de un filtro representando una técnica de interacción seria “Select By Touching”, el cual es representado a continuación haciendo visible sus entradas y salidas, las cuales son definidas como puertos.

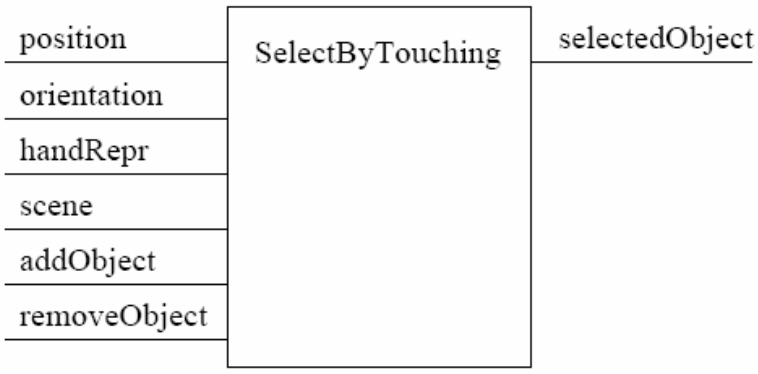


**Imagen 41 Ejemplo de un filtro (SelectByTouching) [FIG 04]**

Como se puede observar en la gráfica los puertos de entrada reciben la información necesaria para que el filtro pueda ejecutarse y realizar su función especifica.

El comportamiento de los filtros está dado por los siguientes pasos:

- Recolección de Datos: Toda la información que llega a los puertos de entrada debe ser recolectada. Este estado también es considerado como un estado de pre–procesamiento en donde la información es agrupada y preparada en cada filtro para su posterior uso.

- Procesamiento: Una vez preparada la información es necesario procesarla generando los nuevos datos que se enviarán al resto del sistema, actividad de la que se encarga este paso.

- Propagación: El proceso final es transmitir la información a cada uno de los elementos que dependen de ella para su procesamiento, esto se realiza enviándola por los puertos de salida.

Los comportamientos de los filtros en los que se basa INTML son utilizados en el modelo de ejecución. Este modelo es lineal y está conformado por los cuatro pasos que se observan en la gráfica (Imagen 42) y que se describen a continuación.

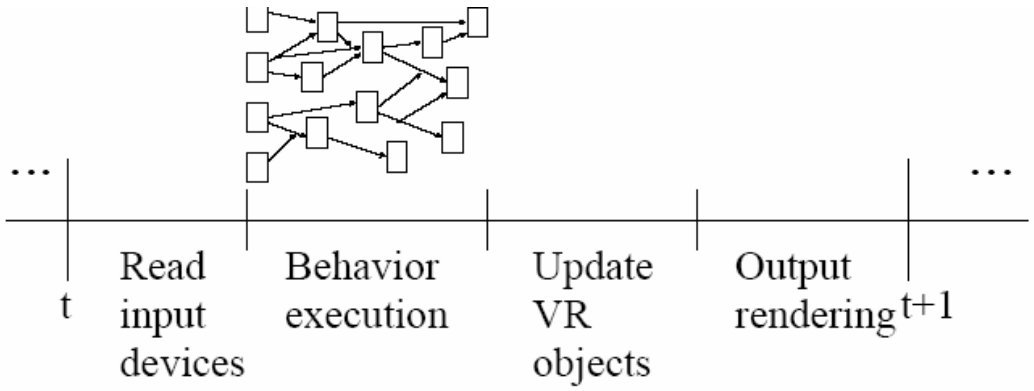


**Imagen 42 Modelo de Ejecución de INTML [FIG 04]**

Los pasos del modelo anterior son los siguientes:

- Lectura de dispositivos de entrada: Este paso es el encargado de obtener la información de cada uno de los dispositivos de entrada registrados.

- Ejecución de flujo de datos: En este paso, se buscan las dependencias de los filtros y cada uno reacciona de acuerdo a los datos recibidos, por medio del comportamiento anteriormente descrito.

- Actualización de los objetos. En el caso especifico en que un filtro represente un objeto, éste debe modificar su geometría de acuerdo a las entradas recibidas, acción de la que se ocupa este paso.

- Render de los objetos: Los cambios que se realizan en la geometría de cada uno de los objetos deben ser visible al usuario, por lo cual el proceso de "renderizado" es el último paso y genera un cambio en la visualización de la aplicación.

Todos los filtros están relacionados con una serie de elementos que conforman el framework para cada plataforma. A continuación se muestra el diagrama general de las posibles relaciones del framework y la descripción de las relaciones entre los diferentes elementos.

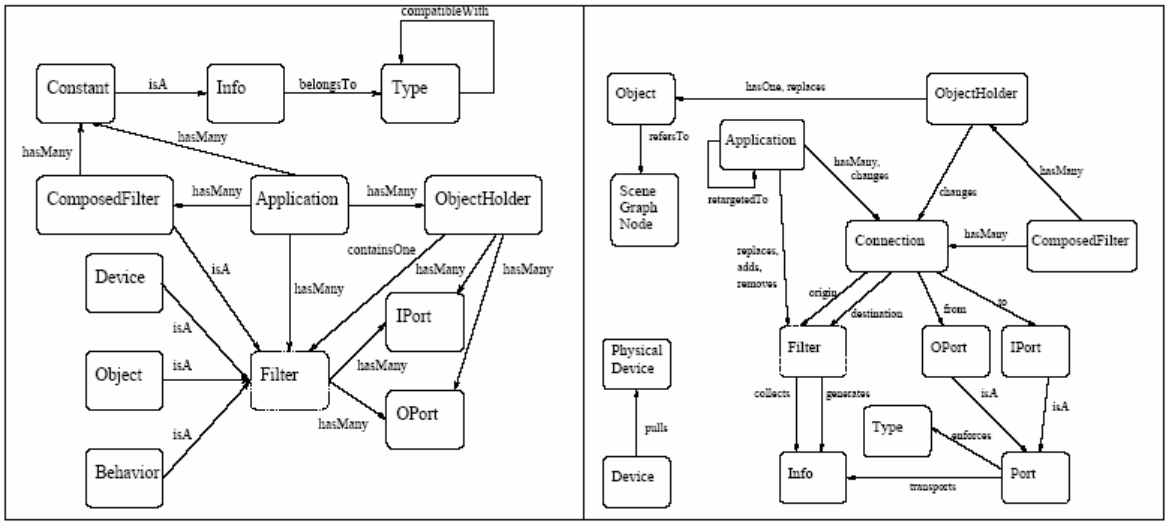


**Imagen 43 Entidades y Relaciones en INTML [FIG 04]**

Por otra parte, INTML busca separar el desarrollo de una aplicación de RV en dos partes por medio de su arquitectura.

- Como primera parte se debe diseñar una aplicación con la estructura de filtros y conexiones que esta metodología propone en el lenguaje que proporciona INTML para esto.

- La segunda parte es el desarrollo de los filtros definidos para un sistema con unas configuraciones específicas, de acuerdo al lenguaje con el que se desea implementar la aplicación.

La anterior arquitectura se muestra en la gráfica a continuación (Imagen 44):

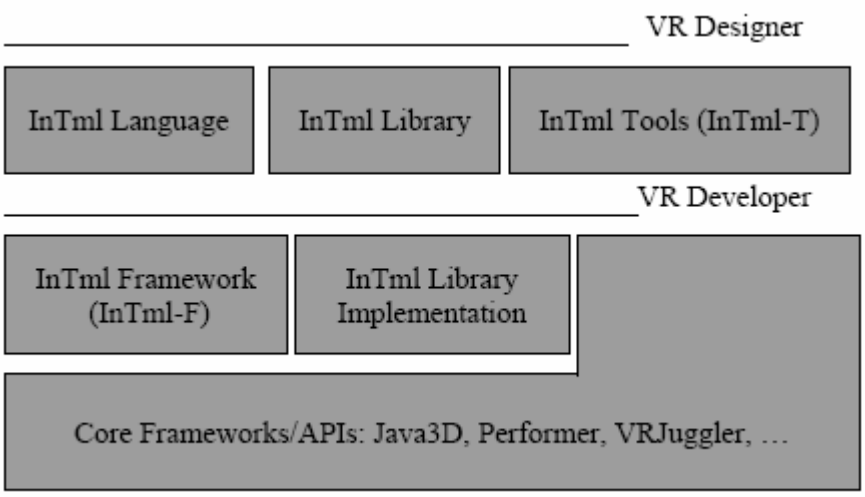


**Imagen 44 Arquitectura de una implementación INTML [FIG 02]**