LIBRERÍA PARA EL MANEJO Y CONFIGURACIÓN DE DISPOSITIVOS DE REALIDAD VIRTUAL

RUBÉN DARÍO GARCÍA CASTILLO

CARLOS ERIC MAHECHA ROA

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA DE SISTEMAS

BOGOTA D.C.

2005

LIBRERÍA PARA EL MANEJO Y CONFIGURACIÓN DE DISPOSITIVOS DE REALIDAD VIRTUAL

RUBÉN DARÍO GARCÍA CASTILLO

CARLOS ERIC MAHECHA ROA

Proyecto de grado para optar el título de Ingeniero de Sistemas

Director:
Ingeniero César Julio Bustacara Medina, MsE.
Ingeniero de Sistemas, Magíster en Ingeniería Eléctrica

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA DE SISTEMAS
BOGOTA D.C.
2005

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA DE SISTEMAS

Rector Magnífico

Padre Gerardo Remolina Vargas S.J.

Decano Académico Facultad de Ingeniería

Ingeniero Francisco Javier Rebolledo Muñoz

Decano del Medio Universitario Facultad de Ingeniería

Padre Antonio José Sarmiento Nova S.J.

Director Carrera de Ingeniería de Sistemas

Ingeniera Hilda Cristina Chaparro López

Director Departamento de Ingeniería de Sistemas

Ingeniero Germán Alberto Chavarro Flórez

Nota de Aceptación:

_____

_____

_____

_____

Director del proyecto

_____

Ing. CESAR JULIO BUSTACARA MEDINA

Jurado

_____

NOMBRE: Ing. Oscar Javier Chavarro

Jurado

_____

NOMBRE: Ing. David Alejandro Uribe

Bogotá D.C., 05 de Julio de 2005

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado.

Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

“Queremos agradecer a nuestras familias por el apoyo y motivación que nos brindaron durante la carrera y en especial al Ingeniero Cesar Julio Bustacara Medina por su orientación y dedicación durante el desarrollo del proyecto ”

“También queremos reconocer la colaboración de Jaime Flautero y Fabio Sánchez en el proceso de pruebas del driver para el casco VFX3D”.

Los autores

TABLA DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

Al iniciar la carrera de ingeniería de sistemas, se pensaba en la interacción que se tiene con el computador, únicamente a través de los videojuegos. Esa constante afición inició el interés de la creación de los mismos, reconociendo varios aspectos que la conforman, como los componentes gráficos, el manejo de los sonidos, la detección de colisiones entre los objetos, la captura de movimientos y la realidad virtual entre otros.

Después del desarrollo de un proceso académico, surgieron nuevas expectativas que se ven reflejadas en esta investigación, partiendo del hecho de los juegos, se piensa en la realidad virtual como una parte importante dentro ellos, pero también aplicable a otros campos como la medicina, comercio, arquitectura y el entretenimiento.

Al conocer los diferentes aspectos que conforman la realidad virtual se adentró en el problema de acoplar mundos virtuales con dispositivos de entrada físicos, en especial en un lenguaje de programación como Java, que regularmente no es utilizado para aplicaciones gráficas y para administrar periféricos de hardware, por esta razón y por interés personal se empezó el desarrollo de esta investigación.

El presente documento esta estructurado con el objetivo principal de establecer las bases teóricas que fundamentaron la investigación, por lo tanto en el capitulo 3 se presentan los conceptos más relevantes de la realidad virtual, haciendo especial énfasis en el hardware y los dispositivos de entrada.

Una vez descritos los conceptos teóricos, en el capitulo 4 se describe el proyecto mediante la definición de las fases y actividades que lo conformaron, posteriormente en el capitulo 5 se describe en detalle el proceso de investigación, de construcción de la librería de administración y configuración de dispositivos para realidad virtual.

En el capitulo 6 se describen todos los aspectos asociados con el plan de pruebas, su ejecución y los resultados obtenidos. Específicamente se describen los procesos que se llevaron a cabo para validar el funcionamiento y desempeño de la librería desarrollada y el análisis realizado en dichos procesos.

En el capitulo 7 del documento se explican los procesos de instalación, se brinda una guía para desarrollar aplicaciones con la librería implementada, en donde se encuentran ejemplos de código Java y los diagramas de actividades que describen las principales tareas que deben realizar los desarrolladores. En la parte final del capitulo se describen las actividades necesarias para adaptar otros tipos de dispositivos de entrada u otras referencias de periféricos similares a los que posee la Universidad Javeriana.

Finalmente en los capítulos 8 y 9 se presentan las principales conclusiones y recomendaciones del proyecto, las cuales surgen del desarrollo y construcción de la librería y del análisis de las pruebas realizadas.

2. GLOSARIO

- *Portabilidad*: Característica que permite a un programa ser fácilmente llevado a otra plataforma o sistema operativo.
- *Plataforma*: es el conjunto de elementos que conforman la base, ya sea de hardware o software, sobre el cual un programa puede ejecutarse.
- *Dispositivo*: Mecanismo o artificio dispuesto para producir una acción prevista.
- *Librería*: es un conjunto de procedimientos y funciones (subprogramas) agrupadas en un archivo con el fin de ser aprovechadas por otros programas.
- *Driver*: Pequeño programa cuya función es controlar el funcionamiento de un dispositivo del computador bajo un determinado sistema operativo.
- *Prototipo*: se puede referir a cualquier tipo de máquina en pruebas, o un objeto diseñado para una demostración de cualquier tipo. Este tipo de prototipos permiten probar el objeto antes de que entre en producción, detectar errores, deficiencias, etcétera.
- *Aplicación*: son los programas con los cuales el usuario final interactúa, es decir, son aquellos programas que permiten la interacción entre el usuario y la computadora. Esta comunicación se lleva acabo cuando el usuario elige entre las diferentes opciones o realiza actividades que le ofrece el programa.
- *Transparencia*: Cualidad de software que le permite operar en distintas plataformas de hardware
- *IDE*: Integrated development environment, es decir, un entorno integrado de desarrollo.
- *Lenguaje*: Cualquier sistema de signos destinados a la comunicación entre seres humanos, entre hombres y máquinas, o entre máquinas, que se estructura sobre un conjunto definido de reglas, convenciones y representaciones gráficas y/o fonológicas.
- *Compilador*: Programa que traduce los códigos fuente en lenguaje de máquina, para que ésta los ejecute.
- *SDK*: Kit de desarrollo de software, un conjunto de aplicaciones para desarrollar programas en un determinado lenguaje o para un determinado entorno (Software Development Kit).
- *Inmersión*: Ver sección 3.1.2
- *Interacción*: Ver sección 3.1.2
- *Tridimensionalidad*: Ver sección 3.1.2
- *Multimedia*: es un término empleado para describir diversos tipos de medios (media) que se utilizan para transportar información (texto, audio, gráficos, animación, video, e interactividad).

- *Tiempo real*: Término que se refiere a un proceso en el que el usuario no percibe el tiempo empleado en su ejecución.
- *VRML(Virtual Reality Modeling Language)*: formato de archivo normalizado que tiene como objeto la representación de gráficos interactivos tridimensionales; en el que se especifican los vértices y las aristas de cada polígono, además del color de su superficie.
- *Cinestésica*: es la capacidad para usar todo el cuerpo en la expresión de ideas y sentimientos, la facilidad en el uso de las manos para transformar elementos. Incluye habilidades de coordinación, destreza, equilibrio, flexibilidad, fuerza y velocidad, así como la percepción de medidas y volúmenes.
- *Exoesqueleto*: Recubrimiento que envuelve el cuerpo, permite transmitir valores de acuerdo a los sensores que se generen en cada movimiento.
- *Rasterizado*: Proceso por el cual una primitiva es convertida a una imagen de 2 dimensiones.
- *Primitiva*: punto, línea, polígono, píxel, mapas de bits.
- *Renderizado*: Proceso mediante el cual el computador crea una imagen partiendo de la descripción de las características de los objetos que contiene.
- *Función de mapeo*: Proceso mediante el cual se transforman valores físicos desde un dispositivo de entrada, a los valores lógicos que representan su estado actual, Por ejemplo una función de mapeo puede tomar valores de entrada desde un puerto serial dados en bits y transformarlos en valores de movimiento dados en grados o radianes.
- *Grados de libertad*: Ver sección 3.1.2
- *CAVE*: Cave Automatic Virtual Environment. Herramienta de visualización, del tamaño de una habitación, que crea en uno o más usuarios la ilusión de estar inmersos en un ambiente virtual. Combina alta resolución, proyección estereoscópica y gráficos tridimensionales.
- *Transductor:* dispositivo que convierte un tipo de energía en otra, para el caso de los dispositivos físicos de entrada puede decirse que son elementos que convierten la energía producida al realizar cualquier tipo de movimiento en valores numéricos dados en bits de información.

3. MARCO TEÓRICO

Dentro del desarrollo de esta investigación se exploraron varios conceptos fundamentales para establecer los horizontes del proyecto y poder elaborar la solución que satisface los objetivos planteados por los investigadores.

En esta sección se presentan tales conceptos, como la definición de realidad virtual, sus componentes (software, hardware), especificación del hardware de acuerdo a los elementos necesarios para la relación entre el usuario y el entorno virtual (estereoscopia, sonido, manejo de objetos, desplazamientos).

También se describen los dispositivos de realidad virtual (cascos, guantes) ya que es la parte más importante de la investigación, las librerías gráficas y de comunicación o medios para lograr la comunicación con los dispositivos.

Estos temas son importantes para poder reconocer todas las variables que están implícitas en la investigación, facilitando el entendimiento y detectando las posibles problemáticas al momento de desarrollar el proyecto.

3.1. Realidad Virtual

Es un área que ha surgido como una evolución de la computación gráfica y que aún se encuentra en desarrollo; una de las definiciones más claras sobre realidad virtual es:

Realidad Virtual (RV) *es una simulación tridimensional interactiva por computador en la que el usuario se siente introducido en un ambiente artificial, y que lo percibe como real basado en estímulos a los órganos sensoriales* **[14]**.

Hoy en día la realidad virtual tiene aplicaciones en sectores empresariales y de negocios, medicina, simulaciones en tiempo real y educación.

3.1.1. Tipos de Realidad virtual

La realidad virtual se puede catalogar en dos grandes grupos:

• Realidad Virtual de Escritorio (No inmersiva): el usuario interactúa con el ambiente sin emplear dispositivos de realidad virtual (guantes de datos, HMD), es decir que la única forma de interactuar con el mundo es a través del mouse y del teclado observando el entorno por medio del monitor.

• Realidad Virtual Inmersiva: son aquellos sistemas en los cuales el usuario observa únicamente el ambiente virtual e interactúa con todos los elementos existentes en dicho ambiente.

3.1.2. Elementos[15]

Los sistemas de realidad virtual se dirigen principalmente a los sentidos (visual, auditivo, táctil) por medio de elementos externos (Cascos de Visualización, Guantes de Datos, Cabinas). De este modo, el usuario puede creer que realmente está viviendo situaciones artificiales que el sistema computacional genera. Las características básicas e imprescindibles son:

- **Inmersión**: Esta palabra significa bloquear toda distracción y enfocarse selectivamente solo en la información u operación sobre la cual se trabaja. Posee dos atributos importantes, el primero de ellos es su habilidad para enfocar la atención del usuario, y el segundo es que convierte una base de datos en experiencias, estimulando de esta manera el sistema natural de aprendizaje humano (las experiencias personales).

- **Interacción**: Rasgos que permiten al usuario manipular el curso de la acción dentro de una aplicación de realidad virtual, permitiendo que el sistema responda a los estímulos de la persona que lo utiliza; creando interdependencia entre ellos. Existen dos aspectos únicos de interacción en un mundo virtual. El primero de ellos es la navegación, que es la habilidad del usuario para moverse independientemente alrededor del mundo. Las restricciones para este aspecto las coloca el creador del software, que permite varios grados de libertad, si se puede volar o no, caminar, nadar.

  El otro punto importante de la navegación es el posicionamiento del punto de vistas del usuario. El usuario se puede mirar a sí mismo (a través de los ojos de alguien más), o puede moverse a través de cualquier aplicación observando desde varios puntos de vista.

  El otro aspecto de la interacción es la dinámica del ambiente, que no es más que las reglas de cómo los componentes del mundo virtual interactúan con el usuario para intercambiar energía o información, dependiendo de las acciones generadas ya sea por parte del sistema o por parte del usuario.

- **Tridimensionalidad:** Antes de hablar de la tridimensionalidad como tal, se debe explicar la definición de espacio, espacio absoluto o relativo; se concibe el espacio en el sentido absoluto, conforme los presupuestos teóricos con los cuales operamos la mayor parte del tiempo (Física de Newton y Geometría Euclidiana). El espacio relativo o relacional, se basa

en la experiencia que se tiene con los elementos perceptibles dentro del campo de visión, las relaciones entre si y con el cuerpo **[47]**.

De acuerdo a las definiciones del espacio absoluto y relacional, se puede tener el concepto de tridimensionalidad, ya que se integran estas dos definiciones para poder generar esta característica dentro de un sistema de realidad virtual.

La tridimensionalidad tiene que ver directamente con la manipulación de los sentidos del usuario, principalmente la visión, para dar forma al espacio virtual, de acuerdo a la representación en un plano cartesiano, agregando la iluminación, para establecer cuales son las partes visibles de los objetos dentro de la escena; los componentes del mundo virtual se muestran al usuario en las tres dimensiones del mundo real, en el sentido del espacio que ocupan, y los sonidos tienen efectos estereofónicos (direccionalidad).

Al mostrar al usuario el mundo virtual en sus tres dimensiones, se mostrarán las transformaciones gráficas de los objetos dentro de la escena, esto se basa en los grados de libertad rotacionales o traslacionales, valores que están dentro de los limites de la capacidad de movimiento real que tienen los seres humanos.

Al mencionar los efectos sonoros, se maneja la sensación de tridimensionalidad, ya que el oído humano, permite ubicar las fuentes sonoras de acuerdo a los tiempos en que se tarda el sonido en ser percibido **[48]**.

3.1.3. Componentes

Dentro de la realidad virtual al igual que en otras áreas de la computación son necesarios componentes de hardware y software, que hacen posible que los elementos mencionados en el numeral anterior puedan aparecer y puedan generar ambientes de realidad virtual.

3.1.3.1. Software **[16]**

Las características más comunes del software de RV incluye la importación de modelos 3D, uso de librerías (gráficas, sonido), operaciones geométricas sobre los objetos, detección de movimiento, manejo de luz, colores y texturas de los objetos.

Un programa de Realidad Virtual presenta una estructura de mayor complejidad que lo normal de un software computacional, debido a que, por un lado, constituye

un ambiente tridimensional que se extiende hasta capacidades multimediales, a la vez que dispone de una programación específica y el control de dispositivos externos, todo funcionando y modificándose en tiempo de interacción. Es necesario administrar aspectos como la entrada y salida de datos, la manipulación de objetos (su estado, jerarquía, geometría) a través de las bases de datos.

En el mercado existe diferentes software para desarrollo de aplicaciones de RV. Todos los productos se caracterizan por diversas opciones para manejar el color, la luz, el sonido, las cámaras, las texturas y otras cualidades. Cabe señalar que actualmente la mayoría de los productos están orientados a Internet, bajo el estándar VRML.

Dentro de las aplicaciones para RV se encuentra a: RayDream Studio, 3D Open System, Vrealm, Internet Space Builder, SitePad Pro.

3.1.3.2. Hardware

Los sistemas de Realidad Virtual se han popularizado por los cascos y guantes, que otorgan la sensación de estar inmersos en los modelos computacionales, y permiten interactuar con éstos.

Los dispositivos de Realidad Virtual se basan en diversas técnicas, pero en general, implican una mayor relación entre el usuario y los ambientes digitales, estos dispositivos se abordan con más detenimiento en la sección de Dispositivos de Realidad Virtual debido a la importancia en la investigación.

Para lograr una buena relación entre los usuarios y el ambiente virtual es necesario tener en cuenta los siguientes elementos **[15]**:

- **Estereoscopía**

Uno de los factores importantes para tener una sensación tridimensional es la visión en profundidad, que permite percibir el volumen de los objetos y la espacialidad del ambiente virtual. Para esto, además de la representación en perspectiva de los modelos, es relevante incorporar la visión doble (estereoscópica). Esto se refiere a la diferencia entre las imágenes del ojo izquierdo y derecho, que son enviadas por el cerebro en una imagen tridimensional única, y se puede reconocer al taparse un ojo e intentar confirmar con la mano la posición de elementos cercanos **[17]**.

Una técnica para lograr una visión estereoscópica consiste en dividir la pantalla en dos partes y colocando una “capota” sobre el monitor con un soporte para la visión fija del usuario. Es una técnica económica y fácil de implementar, se realiza mediante la incorporación de un accesorio para configurar la pantalla, se define el campo de visualización del modelo virtual con dos puntos de vista ligeramente

separados. Sin embargo presenta problemas debido al formato y a la posición rígida del usuario que impide el manejo de otros dispositivos interactivos **[18]**.

Otra técnica que se ha consolidado en instalaciones industriales son los lentes de obturación *(shutter glasses*), que consisten en gafas con cristal líquido que electrónicamente oscurecen cada ojo, coordinados con el barrido de la imagen del monitor, con un secuenciador de la señal entre la pantalla y el computador. De este modo se presenta consecutivamente la imagen izquierda y derecha, y las gafas permiten la visión respectiva obteniendo la visión en profundidad **[18]**.

Figura 1 Lentes de obturación[49]

Entre los sistemas para percibir mundos virtuales se encuentran los cascos o pantallas montadas en la cabeza (HMD: *head-mounted display*s). Dos pequeños monitores transmiten directamente la imagen izquierda y derecha a cada ojo (figura 2), y el usuario se puede mover manteniendo las pantallas frente a los ojos. Varios dispositivos además ocultan la visión del entorno real por los lados de las gafas (oclusión), por tanto el usuario tiene una sensación más fuerte de inmersión en el modelo computacional. Sin embargo este mismo aspecto es el que produce cansancio ocular y mareos, por lo que algunos cascos (denominados "semi-inmersivos") han optado por conservar una visión parcial del ambiente real, para orientar al observador. Las pantallas actualmente alcanzan una resolución y proporción similar a los monitores de computador, pero en dimensiones reducidas (aprox. 2") y colocadas a pocos centímetros de los ojos **[17]**.

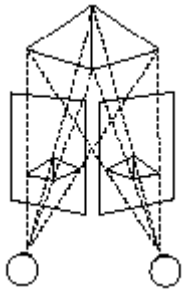


Figura 2 Mecanismo utilizado en HMD para producir estereoscopía

Existen otras técnicas para lograr la visión estereoscópica, como los estereogramas, donde presentan las dos imágenes (figura 3), o par estéreo, pero la imagen izquierda, se coloca de lado derecho y viceversa **[46]**.

Figura 3 Estereograma[46]

El filtrado por color tiene una rejilla de difracción (figura 4) que parece una mica común y corriente, pero funciona de manera semejante a un prisma de cristal, la luz que la atraviesa, se desvía de manera distinta según su color. Cuando uno usa una mica de difracción en un ojo, los objetos parecen tener una profundidad distinta según su color.

Figura 4 Lentes de difracción[46]

En un dibujo el azul se vera siempre en el fondo, el amarillo en medio y el rojo mas cerca. Las imágenes preparadas para este sistema pueden verse de manera normal y sólo con los lentes aparecen en 3D, incluso en imágenes que no fueron diseñadas para 3D, pero que se elaboraron con colores intensos. El inconveniente es que la selección de colores es limitada y no funciona bien con fotografías **[46]**.

La técnica de colocar filtros para separar dos imágenes se llama Anáglifo, Si vemos a través de un filtro rojo, los colores verde o azul se ven como negro y si utilizamos un filtro verde, azul o cian, el rojo parece negro, a partir de este principio podemos mezclar dos imágenes (figura 6) y al observarlas con filtros de color

(figura 5) podemos separar las dos imágenes, finalmente si ponemos un filtro distinto en cada ojo, podemos proporcionarle a cada ojo las dos imágenes necesarias para obtener el efecto estereoscópico **[46].**

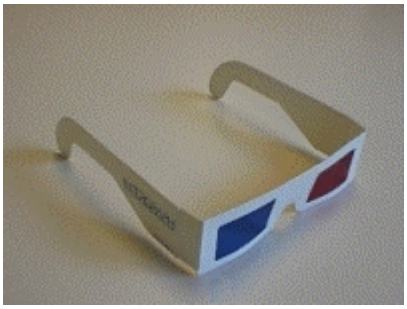


Figura 5 Filtros de color

Figura 6 Mezcla de dos imágenes para observar mediante filtros de color

- **Sonido virtual**

Una de las mayores expectativas de la Realidad Virtual es una representación múltiple de la percepción humana, esto es, involucrando todos los sentidos. Sin embargo, el desarrollo se ha concentrado mayormente en capacidades visuales y cinestéticas. Sólo la reproducción del sonido se ha integrado adecuadamente, basándose en la tecnología multimedia de los computadores actuales**[15]**.

En la mayoría de los sistemas virtuales se pueden incorporar sonidos internos o externos en tiempo-real, estableciendo una posición como fuente para controlar el

volumen según la distancia del usuario, lo que se conoce como “sonido volumétrico” **[17]**.

- **Interfaces Haptics [50]**.

Proviene del griego *Haptai,* que significa tacto, en donde se da una retroalimentación de fuerza y de tacto, se basa en sensores sobre o cerca a la piel, que informa los datos sobre la superficie de contacto, como aspereza, temperatura, bordes y geometría.

La retroalimentación de la fuerza, se basa en la medicion de la fuerza producida por el usuario, en un dispositivo especial, este debe estar apoyado y debe tener un mecanismo mecanico que resista el movimiento del usuario.

Existen varios dispositivos como son el CyberGrasp, CyberForce, LIE, Spider 8, PHANToM.

- **Olor, gusto**

Dentro de los constantes estudios, la compañía Sony Electronics patentó un sistema que permitirá disfrutar del cine con los cinco sentidos. Se encuentra en fase de desarrollo, dicho dispositivo emitirá ultrasonidos encargados de evocar diferentes sensaciones en el cerebro de los espectadores. De esta manera, mientras miremos una película, creeremos percibir aromas, sabores y texturas **[51]**.

- **Desplazamiento tridimensional**

Los dispositivos de salida de datos del mundo virtual se complementan con sistemas de entrada de datos, que permiten desarrollar una actividad en el entorno digital. La acción más básica es el desplazamiento del punto de vista, estableciendo un recorrido por el modelo computacional. Para definir el desplazamiento 3D, lo más sencillo es utilizar el mouse, de que dispone la mayoría de los computadores personales (también se puede utilizar el teclado, pero no se puede ver adecuadamente al utilizar casco). Los joystick se pueden configurar para el desplazamiento en el ambiente virtual de una manera similar, utilizando también los botones o disparadores incorporados. La ventaja es que el movimiento principal (adelante-atrás y derecha-izquierda), con el joystick es más intuitivo y sensible, además se puede sostener con la otra mano o adosar en un lugar fijo, liberando más la acción corporal. Estos dispositivos simples también se pueden configurar para visualizar objetos, moviéndose en torno a éstos, es decir, en vez de las traslaciones en los ejes, privilegiar los giros **[18]**.

Algunos dispositivos manuales, como el *SpaceMouse, SpaceBall* de Microsoft, o Joystick 3D poseen el control inmediato de los seis grados de libertad, a través de un elemento que “flota” en todos los sentidos con gran sensibilidad y a la mínima presión. Estos dispositivos exigen un proceso de aprendizaje, un acostumbramiento del usuario y otorgan una libertad de movimiento que tiende a desorientar. Siempre es recomendable restringir el desplazamiento o actuación al tipo de actividades por realizar para no distraerse excesivamente **[15]**.

- **Manipulación de objetos**

Una relación más completa del usuario con el ambiente virtual exige dispositivos más sofisticados, en los cuales se controle más de un elemento y sus opciones asociadas. Normalmente, éstas se refieren a instrumentos (bastones, manubrios, máquinas o herramientas) o a partes corporales (manos, piernas) **[17]**.

Para los instrumentos virtuales normalmente se reproducen los elementos físicos asociados y se complementa con una modelación computacional del instrumento. El elemento físico y su representación virtual (avatar) se establecen como las referencias fijas en el medio real y computacional. Por ejemplo, en una instalación de manejo virtual la operación se localiza exclusivamente en el instrumental, el usuario y el vehículo virtual se desplazan conjuntamente en el ambiente computacional. En algunos casos, partes del instrumental se mueven indistintamente y controlan acciones diferenciadas (como los bastones y las tablas en un “esquí virtual”), pero deben mantener una conexión física en la instalación y una relación geométrica determinada en la representación digital. De hecho, el usuario puede utilizar una visualización inmersiva con rastreo de orientación, pero el control se realiza en el instrumental asociado al punto de vista virtual **[19]**.

Uno de los dispositivos más comunes y conocidos son los “guantes”, sin embargo han presentado un escaso desarrollo y aplicación por su sofisticación. Aunque la movilidad de la mano permite una gran versatilidad de acciones, la mayor parte ha sido posible representarla con controles específicos (como el mouse y los botones). Estos dispositivos utilizan básicamente dos técnicas, los “exo-esqueletos” y los “guantes de datos”. Este último dispositivo está compuesto de lycra con cables de fibra de vidrio por cada dedo. Cada fibra posee un emisor de luz al inicio y un sensor al final, de modo que se determinan los giros por la intensidad de luz recibida. Posee bastante flexibilidad y portabilidad, pero la identificación de la posición debe realizarse con un rastreador adicional **[15]**.

3.2. Dispositivos de Realidad Virtual

A continuación se abordan algunas definiciones descriptivas de los dispositivos, sus relaciones con el ser humano, características propias de los dispositivos y su evolución dentro de la historia que muestra la importancia de los mismos en la realidad virtual.

3.2.1. Evolución de los dispositivos de realidad virtual

El inicio de la realidad virtual se dá con los primeros experimentos para desarrollar hardware de inmersión, dichos experimentos fueron realizados en la Universidad de Harvard, seguidos de estos estudios Ivan Sutherland (alumno prestigioso de MIT), tomo estos estudios y creó la “espada de damocles” un casco (figura 7) (incredible helmet) que contenía unos tubos de rayos catódicos, con una barra rígida en la parte superior del mismo que permitía traducir los movimientos de la cabeza a unos potenciómetros y llevarlos al computador. Este casco tenía algunos defectos, sobretodo en la movilidad y en su incomodidad.

Figura 7 Incredible Helmet

Las limitaciones del hardware existente en aquella época ocasionaron que las primeras tecnologías resultaran poco convincentes. Los ambientes virtuales se creaban usando el sistema de generación de gráficos vectoriales más avanzado del momento, sin embargo, solo se logró producir la sensación de estar en un mundo de objetos que parecían estar hechos de alambre y la ilusión de inmersión era insuficiente, esto ocurría entre el años de 1966 y 1968 **[10]**.

Después de varios intentos fallidos de crear un ambiente de realidad virtual, las empresas militares y de aviación se involucraron y pudieron crear los primeros simuladores **[7]**. Entre estos se tiene la primera adaptación a un dispositivo que reacciona con respecto a la fuerza aplicada, esto tuvo lugar en 1968 dirigida por Frederick Rooks y constaba de un brazo mecánico con menos resistencia para poder medir y cambiar la fuerza aplicada en el mismo.

Luego de estos primeros avances en aplicaciones y dispositivos viene uno de los personajes más destacados en el campo, como lo es Jaron Lanier, quien creó el guante de datos (**Ver figura 8**) y fundó la empresa VPL Research en compañía de Thomas Zimmerman. Esta empresa empezó a vender su producto a entidades como la NASA y el pentágono creando un mayor interés en la realidad virtual.

Figura 8 DataGlove 5DT

Después de estos avances en los dispositivos y dentro de la realidad virtual, se tiene el adelanto dentro de los productos o aplicaciones que utilizan los cascos, estos productos fueron iniciados en los años 1981 y terminados en 1988, además de la aparición de los primeros juegos multiusuario con interacción en el mismo espacio.

Al momento de la creación de estos productos que involucraban el uso de los dispositivos, se dio lugar a la creación de los lenguajes más favorables para la creación de los entornos virtuales, estos lenguajes fueron creados basados en OpenInventor, por Silicon Graphics que sería llamado VRML (Virtual Reality Modeling Lenguage).

La evolución de la realidad virtual siempre ha estado unida a la evolución de los dispositivos, de esta manera se puede describir el funcionamiento de los mismos en la actualidad. Los cascos (**Ver figura 9**), tienen mecanismos que permiten medir la posición de la cabeza y por ende la del usuario llamados giroscopios; además de brindar al usuario la completa o parcial inmersión, dependiendo del tipo de casco empleado; mediante la utilización de un par LCD se despliegan imágenes iguales para proporcionar la sensación de profundidad y de inmersión para el usuario. **[8]**.

Figura 9 Casco de realidad virtual VFX3D[52]

Los guantes presentan varios sensores dentro de su composición, estos reciben y envían varios tipos de señales, que proveen la información suficiente para conocer que dedo o parte de la mano a tenido un movimiento o cambio de estado.

3.2.2. Dispositivos de entrada

La tecnología de construcción de hardware para realidad virtual ha sido desarrollada hasta un punto tal, que hoy en día existe una gran variedad de dispositivos de entrada con diversas especificaciones, propiedades, diseños y características que hacen necesario establecer parámetros de clasificación que permitan organizarlos de manera tal que se facilite la obtención de una descripción formal de diseño detrás de cada periférico.

3.2.2.1. Definición [26]

"*Básicamente un dispositivo de entrada es un transductor que transforma propiedades físicas del mundo en valores lógicos de una aplicación*"

Formalmente Card**[27]** representa a los dispositivos de entrada como una séxtupla definida como:

(M, In, S, R, Out, W)

En donde:

- **M** es un operador de manipulación
- **In** es el dominio de entrada
- **S** es el estado actual del dispositivo
- **R** es una función mapeo que transforma los valores del conjunto del dominio de entrada a los valores del conjunto del dominio de salida. Dichos valores representan el estado actual del dispositivo.
- **Out** es el conjunto de dominio de salida.
- **W** es un conjunto de propósito general de las propiedades del dispositivo que describen aspectos adicionales acerca de la manera en la que el periférico funciona.

En la figura 10 se listan algunos de los operadores de manipulación posibles para un dispositivo de entrada, en general estas aplican para la mayoría de los dispositivos.

Posicion			
	Absoluta	Posicion P	Rotacion R
	Relativa	Delta Movimiento dP	DeltaRotacion dR
Fuerza			
	Absoluta	Fuerza F	Torque T

Figura 10 Propiedades físicas sensadas por dispositivos de entrada[26]

3.2.2.2. Composición y clasificación de los dispositivos de entrada

Los dispositivos de entrada pueden llegar a ser tan complejos como las tareas que deben realizar, por lo tanto en muchos casos es bastante difícil clasificarlos y sobre todo es complicado conocer sus especificaciones para determinar la mejor manera de abstraer su comportamiento a un dispositivo lógico, por lo tanto es importante tener claro que dichos periféricos pueden ser vistos como la unión de otros dispositivos más simples que cumplen una función específica dentro de un todo, en esta sección se describirán las diferentes formas de composición de dispositivos y el espacio de diseño que permite clasificarlos de acuerdo a sus características físicas (taxonomía).

3.2.2.2.1. Operadores de composición [27]

Algunos dispositivos pueden verse como un conjunto de dispositivos más simples conectados de alguna manera, por ejemplo un radio sintonizador de AM / FM puede verse como la composición de otros dispositivos como el control de volumen, el de selección y el display de las estaciones, en donde el dominio de salida del control de selección es el dominio de entrada del display. Existen tres tipos de operadores de composición de dispositivos:

- Composición Merge
- Composición Layout
- Composición Connect

El merge es la combinación de los dominios de entrada de los dos dispositivos. Por ejemplo un mouse puede verse como la composición de dos componentes que finalmente determinan las coordenadas X,Y. El layout es la colocación de dos dispositivos en lugares diferentes de un panel común, en el ejemplo del mouse los dos botones y el sensor de coordenadas componen juntos el dispositivo total. La composición Connect ocurre cuando el dominio de salida de un dispositivo es el dominio de entrada del otro dispositivo. Para el mouse la salida es conectada a la entrada del cursor que se mueve en la pantalla, el cursor no es un dispositivo físico pero puede verse como un dispositivo lógico.

3.2.2.2.2. El espacio de diseño de los dispositivos de entrada [27]

El conjunto de posibles combinaciones entre los operadores de composición recibe el nombre de “Espacio de Diseño” el cual permite describir de una manera sencilla la taxonomía de un determinado dispositivo de entrada, en la figura 11 se describe la taxonomía del mouse:

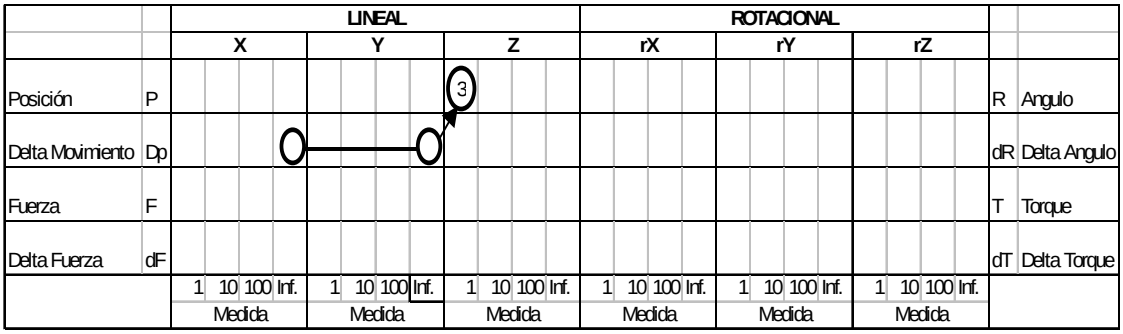


Figura 11 Taxonomia del ratón (mouse)

En la figura 11 los círculos son usados para indicar que el dispositivo posee una propiedad física que corresponde al eje en el que se encuentre tanto horizontal como verticalmente, las líneas son usadas para conectar los círculos de cada componente del dispositivo, si se tiene una línea sencilla se indica un **Merge,** si se tiene una línea doble se tiene un **Connect,** la flecha indica un **Layout.**

Para el caso del mouse se tiene una composición Merge de movimiento en los ejes (X, Y) y una composición Layout con los botones del dispositivo en Z, el número de elementos que conforman el Layout se debe indicar dentro del circulo, en el ejemplo de la figura 11 se manejan 3 botones.

El espacio de diseño es un diagrama que facilita la clasificación de los diferentes dispositivos de entrada de acuerdo a su estructura y a las características propias del periférico, este tipo de diagrama será empleado posteriormente para describir formalmente los dispositivos de entrada empleados en este proyecto de investigación.

3.2.2.3. Aspectos importantes dentro del desempeño de los dispositivos de entrada[27]

Existen variables dentro de los dispositivos de entradas que hacen que el desempeño del periférico pueda variar considerablemente al momento de emplearlo en las diferentes aplicaciones de computación gráfica, básicamente estas variables son:

- Ancho de banda del dispositivo
- Precisión
- Errores de hardware
- Tiempo de aprendizaje
- Tiempo para dominar el dispositivo
- Tiempo de procesamiento de información

Estas variables incluyen medidas de desempeño de acuerdo al usuario, como lo son los tiempos de aprendizaje y dominio del dispositivo, estos tiempos están relacionados, con el esfuerzo para su comprensión (aprendizaje) y la habilidad para manejarlos (dominio), por ejemplo, un niño toma un tiempo para aprender la manera de patear un balón alcanzando solo mover el esférico, sin tener en cuenta características que se deben lograr para que este sea un buen disparo, como precisión ó fuerza; al desarrollar habilidades mediante la práctica, el niño puede mejorar y aplicar las características propias de un buen disparo, el tiempo empleado para el desarrollo de habilidades se denomina tiempo de dominio. También existen variables que dependen del hardware y del software que intervienen en la puesta en marcha del periférico. En esta sección se describirá en detalle el aspecto más importante dentro de un dispositivo de entrada: el ancho de banda.

3.2.2.3.1. Ancho de banda de los dispositivos de entrada

El ancho de banda es definido como la cantidad de información que puede ser obtenida por la aplicación desde los dispositivos de entrada**[28]**. Pero el ancho de banda no solo depende del dispositivo, sino además, de la aplicación y de la anatomía del actor que lo controla. Es decir, existe un ancho de banda del humano, de la aplicación y del dispositivo como tal.

a. Ancho de banda Humano: se limita al ancho de banda del grupo de músculos a los que se encuentra sujeto el dispositivo.

En la figura 12 se muestran los resultados del experimento de Langolf**[29]** con los valores del ancho de banda para el cuello, brazo, muñeca y dedo, dichos valores fueron obtenidos mediante el uso de la denominada ley de **Fitts[53],** la cual describe el tiempo requerido para acceder a un objetivo visual mediante algún tipo de dispositivo de entrada. Entre las diferentes variantes de esta ley la más comúnmente utilizada es:

$$TiempoAcceso = C1 + C2\log_2((W/C) + 0.5)$$

En donde D es la distancia al centro del objetivo, W es el tamaño del objetivo, C1 y C2 son constantes determinadas experimentalmente. El valor de la parte logarítmica de la expresión es denominado *índice de dificultad* (ID) el cual esta dado en **Bits**, por lo tanto la fórmula de la ley de Fitts puede expresarse como sigue:

$$TiempoAcceso = C1 + C2ID$$

El valor **1/C2** es denominado índice de desempeño el cual esta dado en bits por segundo. En el experimento de Langolf los movimientos de diferentes músculos causaron diferentes índices de desempeño los cuales son mostrados en la figura[6].

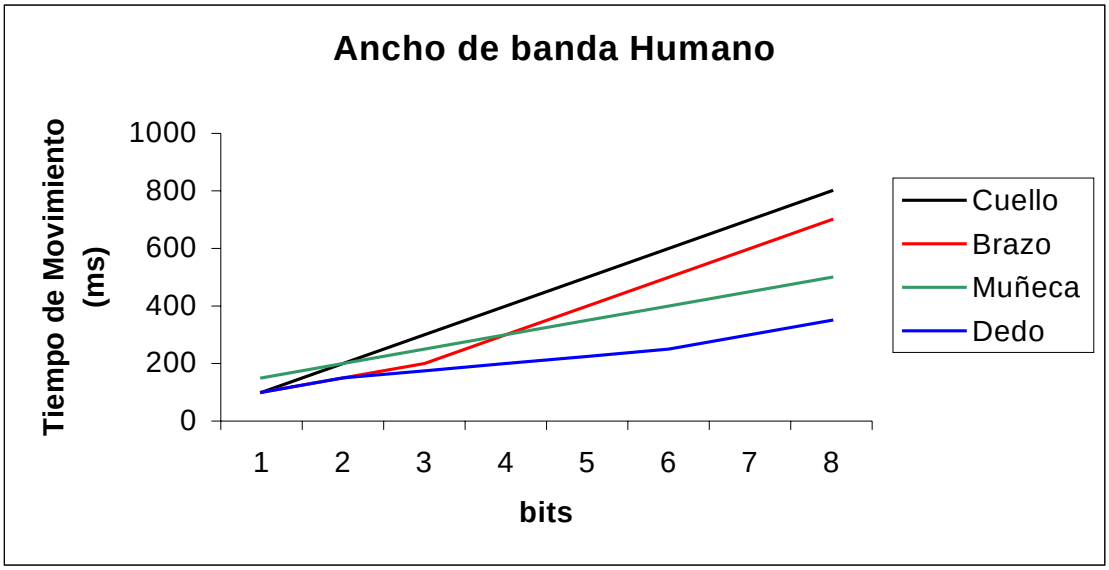


Figura 12 Gráfico de resultados del experimento de Langolf[29]

A raíz de este experimento se obtuvieron los siguientes valores:

Ancho de banda cuello: 5 bits/segundo
Ancho de banda brazo: 10 bits/segundo
Ancho de banda muñeca: 23 bits/segundo
Ancho de banda dedo: 38 bits/segundo

Por lo que se puede concluir que un dispositivo que trabaje sobre el cuello (como un casco) emitirá mucha menos información por segundo que un dispositivo que trabaje con los movimientos de los dedos o de la muñeca (como un guante de datos).

b. Ancho de banda del dispositivo: Es la cantidad de información proporcionada por el dispositivo de entrada físico.

c. Ancho de banda de la aplicación: Son los requerimientos de precisión de la tarea que será realizada con el dispositivo.

3.3. Librerías Gráficas

Las librerías gráficas a mencionar presentan comportamientos y métodos similares, sus principales diferencias se encuentran en el lenguaje, las plataformas en que pueden ser empleadas y en el rendimiento.

3.3.1. OpenGL

OpenGL ha tenido varias versiones desde la 1.1 hasta la 2.0, sin embargo las diferencias no son considerables y no cambian en su empleo y manejo. OpenGL es una interface para desarrollos gráficos dependiendo del hardware.

Esta interface consiste en un conjunto de funciones y procedimientos que permite al programador especificar los objetos envueltos en la definición de objetos de alta calidad, específicamente imágenes de 3 dimensiones a color**[35]**.

OpenGL no contempla el uso de los dispositivos de entrada, como el teclado o mouse, para su correcto funcionamiento el programador debe encontrar otros medios para su manejo.

Para la definición de los objetos presenta la existencia de primitivas base estas son básicamente líneas, puntos, triángulos, círculos, cuadrados que acompañadas de texturas permiten generar los sólidos.

Las primitivas especificadas y las demás operaciones de la librería gráfica, son descritas y enviadas por comandos o por llamadas de procedimientos. Las primitivas son definidas por un grupo de uno o más vértices, el vértice se define como un punto en el espacio, ya sea el comienzo o final de una línea o la intersección de dos lados de un polígono o de las primitivas.

Muchos de los datos necesarios para la construcción de las imágenes, están relacionados con cada vértice, como es la textura, el color y las normales, cada uno con su objetivo específico en la creación del sólido, estos datos son procesados en orden de acuerdo al vértice, aunque tiene algunas excepciones dependiendo de la región específica en la que se encuentren.

Dentro de la librería, se presentan varios estados de las variables involucradas, estos estados están relacionados con procedimientos o llamadas al buffer de entrada o principal (framebuffer). Se distinguen dos tipos de estados, el primer tipo es llamado GL *server* y reside en el GLX server, la mayoría de los estados está dentro de esta categoría y el otro tipo son llamados GL *client state* que esta en el GL client.

En la figura 13 se puede observar como se realiza el proceso de pintado del objeto, mientras se puede estar manejando diferentes objetos en diferentes estados o comandos.

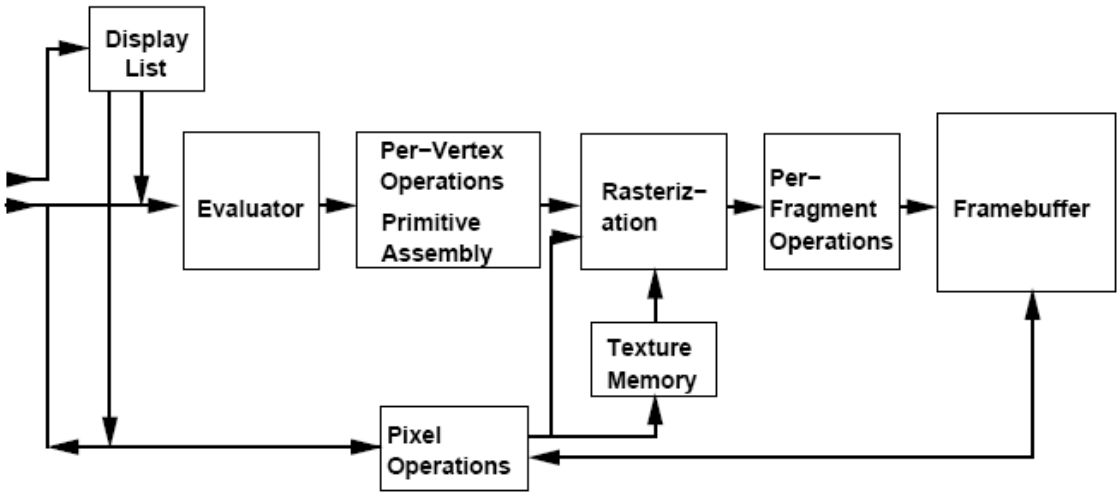


Figura 13 Proceso de pintado del objeto [35]

Cada uno de los vértices tiene un conjunto de coordenadas, además de su normal, su textura, atributos genéricos, color actual, secundario y atributos de sombreado (fog).

Las normales son utilizadas por GL como cálculos de la iluminación del objeto, los datos de texturas determinan como se va a mapear la textura sobre la primitiva que se desea pintar, el color secundario y primario son asociados a cada vértice perteneciente a la figura, de acuerdo a esta asociación se basan los cambios del color efectuados por la luz y también los del sombreado.

La figura 14 describe los mecanismos a realizar con cada uno de los datos, donde se puede presenciar el cambio de un vértice sin procesar a un vértice procesado.

Después de tener el vértice procesado se puede llamar u obtener la primitiva creada a partir de estos vértices, este proceso se muestra en la figura 15.

Dentro de los métodos principales para poder pintar una figura o escena en 3D, se tiene en cuenta el tipo de primitiva a dibujar, ya que al inicio del pintado se debe especificar si es un triangle, quads o polygon, estas son algunas de las posibilidades para pintar los objetos o primitivas dentro de OpenGL.

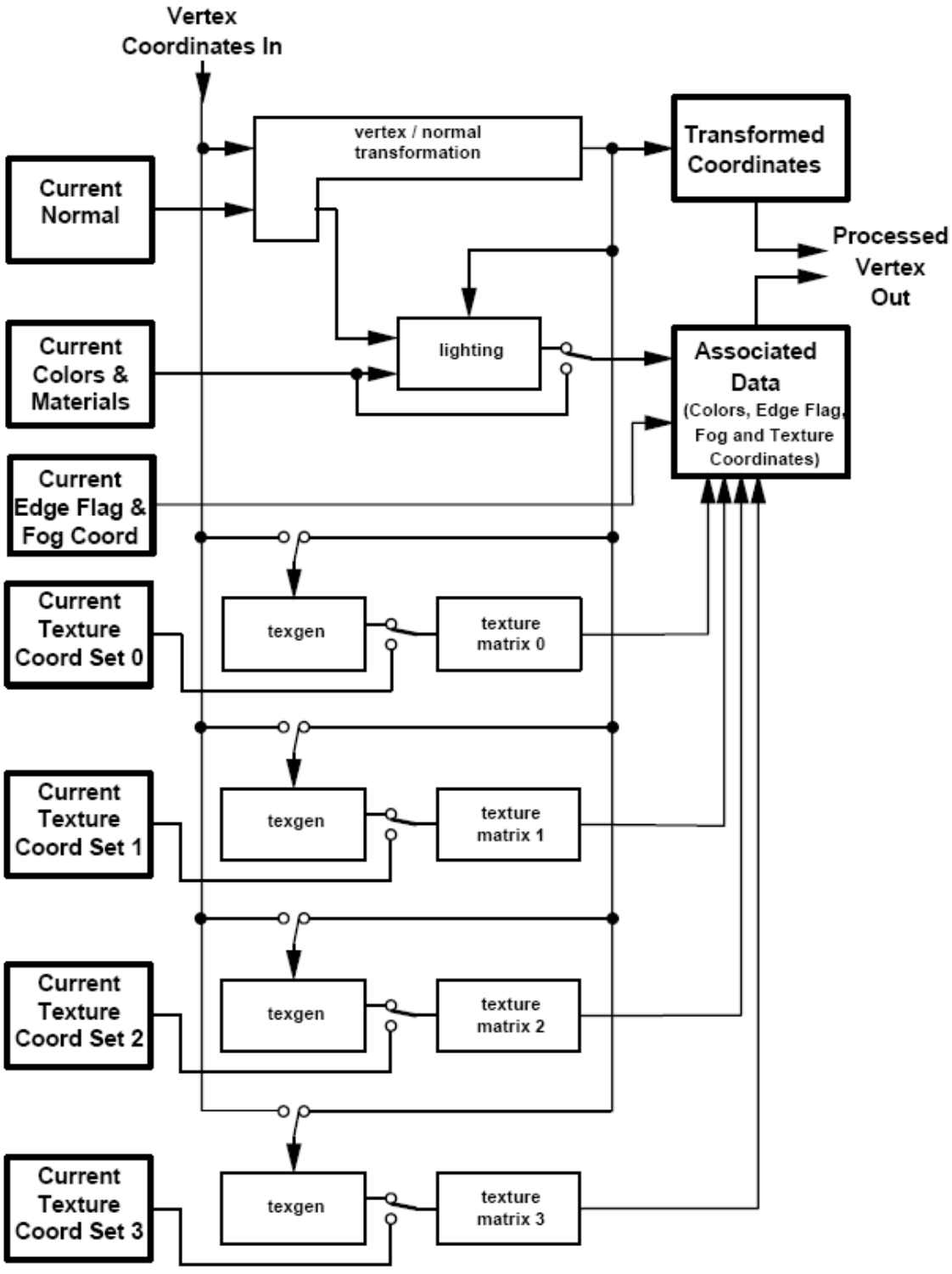


Figura 14. Mapeo de valores de rasterizado [35]

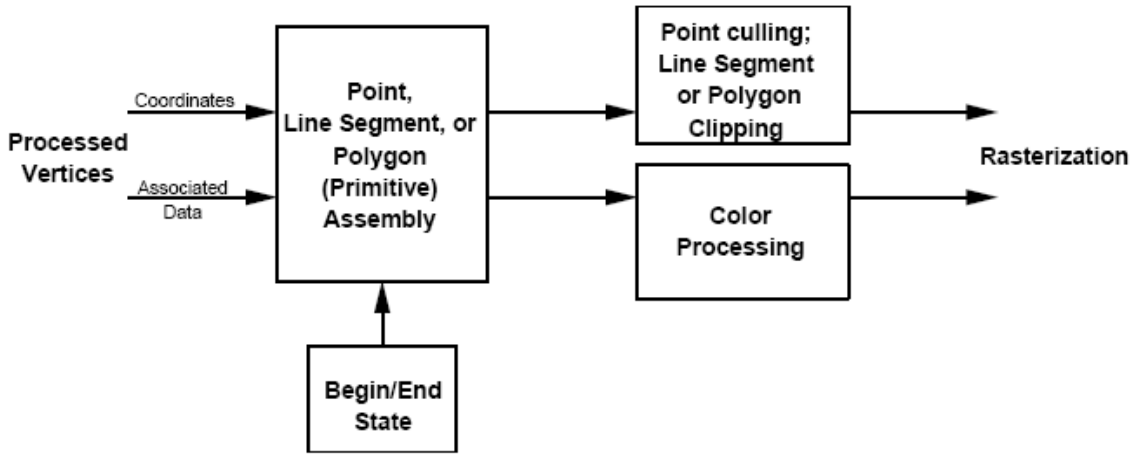


Figura 15. Proceso anterior al rasterizado [35]

En la figura 16 se observa el proceso de rasterización, que permite convertir cada una de las primitivas en una imagen de 2 dimensiones.

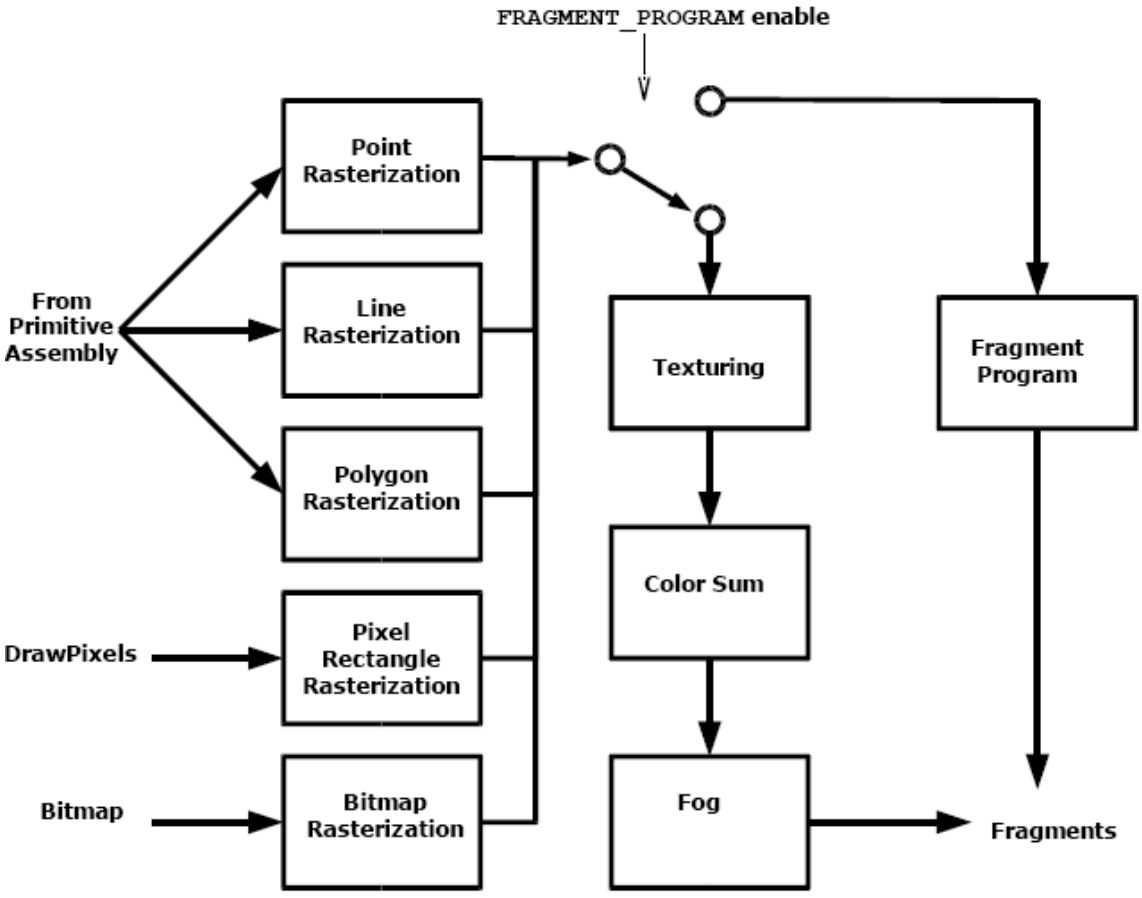


Figura 16 Rasterizado [35]

Después de estos procesos se llega a la realización de los objetos de alta calidad a partir de los objetos ya rasterizados, por medio del renderizado.

En resumen se puede ver como realiza este proceso descrito anteriormente varias veces hasta conseguir la unión de las primitivas que formarán el modelo geométrico deseado, y así poder dibujar escenarios 3D.

OpenGL funciona en sistemas operativos como el OS del Mac, OS/2, UNIX, Windows 95, Windows NT, Linux, OPENStep, Python, y BeOS. OpenGL es accesible desde Ada, C, C++, FORTRAN y Java; ofreciendo independencia de protocolos y topologías de red **[12]**.

3.3.2. DirectX

Es un conjunto de APIs, que permiten el desarrollo de aplicaciones multimediales, entre estos se tienen los ambientes virtuales. Esta librería esta diseñada para funcionar bajo sistemas operativos Windows, las APIs que conforman DirectX, son**[13]**:

- Direct 3D: permite la construcción de figuras tridimensionales o bidimensionales con sus respectivas propiedades (iluminación, texturas, etc)
- DirectSound y DirectMusic: Permite la utilización de archivos MIDI o WAV, para la reproducción de sonidos. Además puede manejar el sonido 3D o envolvente dentro de las aplicaciones.
- DirectPlay: facilita la construcción de aplicaciones multimediales distribuidas, como juegos multiplayer.
- DirectShow: es un reproductor de archivos MP3 y MPEG.
- DirectInput: maneja los periféricos, tanto de entrada y salida, como son el teclado, mouse y joystick.

Dentro del API de Direct 3D, existe varias partes que permite la creación de las figuras, como son el DirectDraw, DirectDrawSurface, DirectDrawClipper, etc. Cada una permite realizar algunos procedimientos para el pintado de las figuras.

- DirectDraw es el objeto básico para todas las aplicaciones. Representa la tarjeta de video. Es el punto inicial dentro de cada aplicación.
- DirectDrawSurface representa el área en memoria, es el que permite mantener una imagen fija o en movimiento sobre otra superficie.
- DirectDrawPalette: representa de 16 a 256 colores, permite indexar cada objeto de acuerdo a la tripleta RGB, que describe el color asociado a cada objeto.

- DirectDrawClipper: Permite prevenir que las aplicaciones se esten dibujando fuera del área predefinida.
- DirectDrawVideoPort: Permite el acceso al frame buffer sin la intervención de la CPU o PCI bus.

3.3.3. Mesa

Mesa es una implementación de código abierto del API gráfico OpenGL fue originalmente diseñado para sistemas Unix/X11, pero también esta disponible para otros sistemas como Amiga, Apple Macintosh, BeOS, NeXT, OS/2, MS-DOS, VMS, Windows 9x/NT.

Para poder utilizar mesa se debe contar con un compilador ANSI C y un ambiente de desarrollo particular.

En cuanto a su funcionamiento Mesa es muy similar a OpenGL hasta el punto que utilizan la misma sintaxis de comandos en la máquina de estados, uno de las características importantes en Mesa es el renderizado denominado "***off-screen"*** el cual facilita la generación de imágenes 3D sin tener que abrir una ventana en la pantalla para realizarlo**[24].**

3.3.4. OpenGL for Java (GL4Java)

GL4Java puede ser visto como una extensión de JAVA, con una parte que utiliza openGL nativo y otra propia en lenguaje JAVA, esta funciona bajo Unix, GNU/Linux, Solaris, Irix, Windows 9x-NT, Macintosh OS 9.Y.Z.

OpenGL for Java, no presenta grandes cambios sobre openGL, la diferencias en cuanto a la construcción de un escenario, se basa en el lenguaje en que se programará el manejo de los diferentes dispositivos de entrada, en este caso java.

Aparte de esta diferenciación, los procesos para el correcto pintado de los objetos es el mismo y en el código las funciones son las mismas, con una pequeña diferencia de sintaxis que es casi imperceptible para el programador.

3.3.5. Java bindings for Open GL (JOGL)

Esta librería fue desarrollada y creada por Sun MicroSystems, por dos desarrolladores llamados Ken Russell y Chris Kline, esta es una de las librerías más robustas creadas para Java, nacida por la necesidad creciente de reunir este lenguaje orientado a objetos con una librería gráfica como OpenGL.

JOGL se basa en la utilización de JNI (Java Native Interface), como mecanismo de comunicación entre los métodos nativos de OpenGL y el lenguaje java, permitiendo tener facilidad en la creación de las funcionalidades y presentando los mismos servicios del API gráfico de OpenGL**[34].**

3.4. Arquitecturas para realidad virtual

Al conocer las principales partes de la realidad virtual como lo es el hardware y el software, se mencionan algunas arquitecturas existentes, explicando sus principales características, partes, ventajas y desventajas, en cuanto al manejo de los dispositivos de entrada y salida.

Las arquitecturas en general deben garantizar la realización de ciertas etapas que demandan gran capacidad de procesamiento. Estas etapas se pueden clasificar como:

1 Renderizado de las imágenes donde se encuentra la geometría y el rasterizado.
2. Manejo de colisiones
3. La parte gráfica, desde el punto de vista de las consecuencias sufridas por las colisiones.

Dentro de las arquitecturas que se van a mencionar se tiene a VRObject, VRJuggler y Java3D, donde se explicarán sus principales aspectos dentro de la tabla 1, además se muestran los ambientes que se pueden crear en Java y sus dependencias, desventajas y ventajas.

VRJuggler permite el manejo de los elementos mínimos de realidad virtual, pero para el desarrollo de esta investigación, se centra en la parte del Output Manager e Input Manager, como se observa en la figura 17, en la parte resaltada.

Arquitectura	**Librería Grafica**	**Lenguaje**	**Dispositivos**	**Desventajas.**	**Ventajas**
VR Object	OpenGL, DirectX, Mesa.	C++			Elementos para interactuar con los dispositivos de entrada y salida.
VR Juggler (figura 9)	OpenGL	C++	Joystick, Teclado, Mouse, Casco, Guantes	Utilización necesaria de los drivers de los fabricantes. No portabilidad.	Robustez, manejo de elementos en tiempo de ejecución, escalabilidad y flexibilidad.

Java 3D	OpenGL, DirectX	Java	Teclado y mouse	Limitación en la portabilidad, solo solaris y windows. Implementación de la interfaz para la comunicación con dispositivos	Realizado en Java. Robusto.

Tabla 1. Arquitecturas para realidad virtual

En la tabla 2 se puede observar cómo se relacionan las demás librerías gráficas con las arquitecturas presentadas, dependiendo del lenguaje de programación que se puede utilizar.

Lenguajes de Programación	OpenGL	Java3D	DirectX	Mesa	OpenGL4Java	JOGL
C	SI	NO	SI	SI	NO	NO
C++	SI	NO	SI	SI	NO	NO
Pascal	SI	NO	SI	SI	NO	NO
JAVA	NO	SI	SI	NO	SI	SI
Visual C++	SI	NO	SI	SI	NO	NO

Tabla 2. Compatibilidad librerías gráficas frente a lenguajes de programación

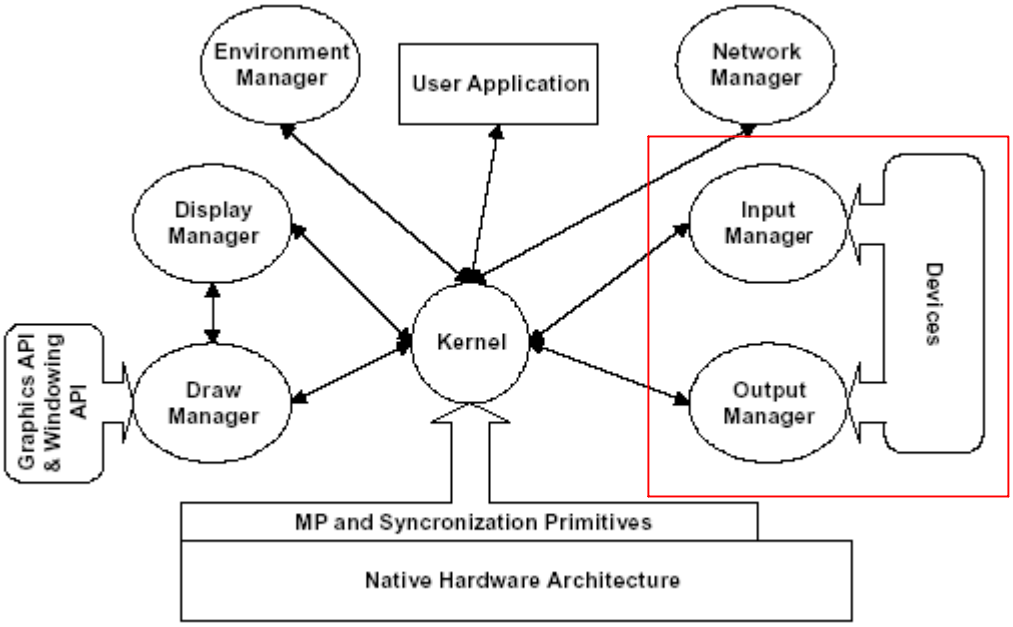


Figura 17. Arquitectura VR Juggler y su elemento principal dentro de la investigación[9].

3.4.1. Arquitecturas de Realidad Virtual en JAVA

Las arquitecturas que son construidas en JAVA, tienen su componente gráfica diseñada e implementada mediante la utilización de openGL4Java, DirectX, JOGL o JAVA3D, pero el manejo de dispositivos hace bastante difícil realizar mundos totalmente inmersivos.

El lenguaje de programación Java ofrece algunos elementos nuevos que permiten mostrar aplicaciones de computación gráfica y de realidad virtual en **Applets** y por lo tanto pueden ser vistas desde Internet, con las limitaciones que esto implica. Pero en general las aplicaciones de realidad virtual en Java pueden llegar a ser comparables con las aplicaciones hechas en otros lenguajes.

Los drivers que se encuentran en la actualidad para el manejo de los periféricos de realidad virtual, están diseñados e implementados en lenguajes como C++ y Visual C++ principalmente, debido a esto, el manejo de los dispositivos con estos lenguajes es más sencillo que en otros, en los que es necesario hacer la implementación para cada uno de los periféricos, dicha implementación requiere la interrupción de puertos y el manejo del flujo de la información, haciendo el procedimiento más complejo y costoso.

De acuerdo a la información citada, se puede establecer la necesidad de crear algún mecanismo que facilite y mejore la comunicación entre los dispositivos y java, para poder aprovechar todas las ventajas que posee Java como lenguaje de programación orientado a objetos.

3.5. Modelos para el manejo de dispositivos de entrada y salida

El manejo de dispositivos de entrada es una tarea crítica dentro del desempeño de una aplicación de realidad virtual ya que la comunicación entre el mundo y el periférico debe garantizar que la información del estado actual del dispositivo lógico corresponda al estado actual del dispositivo físico, por lo tanto debe existir una capa intermedia que realice el mapeo de la información y que obtenga la mayor cantidad de información del periférico, dependiendo del ancho de banda que maneje el dispositivo. Existen diferentes modelos de administración de dispositivos, entre los más importantes se encuentran el modelo de InputDevice de VRJuggler y el modelo GKS (Graphical Kernel System) de la ISO, los cuales serán descritos a continuación:

3.5.1. Modelo GKS

GKS (Graphical Kernel System)**[32]** es el estándar de la ISO**[31]** para la computación gráfica, en donde también se especifica la manera de emplear dispositivos de entrada.

Antes de describir el modelo es importante mencionar un concepto importante dentro del mismo, este es el **valor de medida** (measure value), el cual es definido como el valor de un dispositivo de entrada físico transformado mediante una función de mapeo para ser manejado por el dispositivo lógico**[32].**

El modelo GKS de dispositivos de entrada lógicos puede ser descrito en términos de atributos, clases o tipos de retorno y modos de operación.

- **Atributos**

Son parámetros que permiten al programa de aplicación tener algún grado de control sobre el dispositivo de entrada.

- **Clases o tipos de retorno**

  - **LOCATOR:** Es una posición dada en coordenadas y el número asociado a la transformación de los valores empleados por el dispositivo a las coordenadas que maneja la aplicación.
  - **STROKE:** Es similar a LOCATOR pero este representa una secuencia de posiciones dadas en coordenadas y no solo una ubicación.
  - **VALUATOR:** Es un número real en algún rango.
  - **CHOICE:** Un entero que representa una selección dentro de un conjunto de posible elecciones.
  - **PICK:** El nombre de un segmento seleccionado y un identificador que indica el conjunto dentro del segmento que a sido escogido.
  - **STRING:** Una cadena de caracteres.

- **Modos de operación**

Determinan la forma en que la entrada (valor de medida) es obtenida desde el dispositivo de entrada lógico. Los diferentes modos de operación son:

- **Request:** Una solicitud es hecha por el programa de aplicación para obtener un valor de medida desde un dispositivo especificado. GKS espera hasta que el operador tiene el conjunto de medidas del valor deseado y mantiene activado el trigger.

- **Sample:** En este modo el valor actual de medida es retornado siempre que sea solicitado por el programa de aplicación. Cuando se ejecuta este modo no hay proceso de triggering (ver siguiente sección) de modo que la aplicación continuará inmediatamente después de realizar el llamado al Sample_Mode.

- **Event:** Un número determinado de dispositivos de entrada pueden estar activos simultáneamente en este modo. Cada vez que el trigger de un dispositivo en particular es activado el valor actual de medida y los datos que identifican al dispositivo son agregados a una cola de eventos de entrada para todos los dispositivos que se encuentran en este mismo modo de operación. El programa de aplicación puede consultar a la cola para recuperar los eventos de entrada almacenados allí. Es posible acoplar más de un dispositivo de entrada al mismo trigger de modo que múltiples eventos pueden ser generados desde un solo trigger de eventos. La cola de eventos esta estructurada como una cola de reportes de eventos. La cola de eventos es consultada por la función de GKS **AWAIT_EVENT**, esta función remueve el reporte de la cabeza de la cola y lo escribe en un buffer conocido como "REPORTE DE EVENTO ACTUAL" y devuelve la identificación del dispositivo que produjo el reporte (identificador de la estación de trabajo, nombre de la clase de entrada, número del dispositivo de entrada, etc.) al programa de aplicación. Si la cola esta vacía GKS es suspendido hasta que llegue un evento de cualquier dispositivo (en este caso la función se comporta como antes) ó hasta que el periodo de Timeout expire (en este caso se retorna **NONE**). GKS provee un conjunto de funciones para cada clase de dispositivo en los cuales se retorna el valor de entrada lógico contenido en el "REPORTE DE EVENTO ACTUAL" (Current Event Report).

- **PROCESOS**

Dentro de la especificación de la ISO, GKS se puede definir en términos de seis procesos **[30]**:

- **Medida:** Este proceso mantiene el **valor actual de medida** del dispositivo lógico como el operador que manipula el dispositivo físico

- **Trigger:** Es un evento en el cual para cierto estilo de entrada determina **cuándo** el valor de medida es retornado al programa de aplicación.
- **Prompt:** Este proceso indica al operador cuando el dispositivo esta disponible para obtener y manejar los datos**.**
- **Echo:** Proporciona retroalimentación del **valor actual de medida** al operador.
- **Acknowledgement:** Informa al operador del valor entregado al programa de aplicación.
- **Control:** Controla la operación del dispositivo de entrada lógico.

3.5.2. Modelo VRJuggler

VRJuggler es una aplicación orientada a objetos, que soporta el desarrollo de aplicaciones inmersivas, es independiente del hardware, de la arquitectura y del sistema operativo.

VRJuggler consta de varios elementos que permite la interacción entre los diferentes componentes, entre estos se tiene el JVP (Juggler Virtual Platform), MicroKernel, Kernel, Internal Managers y External Managers.

El JVP se compone a su vez, de las aplicaciones objeto, del manejador de gráficas y del VRJuggler Kernel.

Entre la interfaz de la aplicación y JVP se tiene un elemento llamado el Kernel Interface que provee la abstracción del hardware para las aplicaciones virtuales; además de este elemento se cuenta con el manejador gráfico que provee la abstracción para el manejo de los gráficos, que puede ser implementado usando OpenGL.

Las principales características de JVP son:
- Independiente de la arquitectura y sistema operativo.
- Abstracción de los dispositivos, ya sean posicional, digital y/o guantes.
- Se pueden utilizar varios API gráficos.
- Ambiente de operación, provee un sistema para el manejo de las aplicaciones de realidad virtual.

3.5.2.1. MicroKernel

El microkernel posee dos manejadores, el externo y el interno, el interno esta dirigido al manejo de los dispositivos de entrada, como son las propiedades de la

pantalla o entorno gráfico, la información de configuración y la comunicación con las aplicaciones.

El manejador externo ofrece los mecanismos para la comunicación con el sistema VR Juggler y el Kernel, además contiene el manejador gráfico que controla el API gráfico acompañado con el sistema de sonido.

El VR Juggler Kernel es una arquitectura modular que permite agregar manejadores y reconfigurarlos en tiempo de ejecución. El kernel solo carga los módulos necesarios para la aplicación que se esta ejecutando, esto permite que tenga robustez, ya que puede ejecutarse con cualquier combinación de módulos.

Por ejemplo, este puede ser ejecutado sin ningún manejador, de acuerdo a las necesidades prioritarias, que están definidas según el orden de renderizado, se van cargando los módulos necesarios.

Los manejadores son objetos activos, que se sincronizan por el Kernel, ya que este presenta un uso de multihilos. El hilo principal del kernel puede controlar el tiempo de activación de los objetos del sistema.

El kernel también se puede ver como un mediador (Patrón de diseño), ya que encapsula el control de algunos de los componentes que interactúan en el sistema, esto debido a que el kernel es el único que conoce a los manejadores, dando como resultado la no dependencia entre los mismos. El kernel tiene como tarea la asignación de tiempos de ejecución de algunos de los manejadores, tiene la capacidad de cambiar la manera como el sistema ejecuta los llamados.

Una ventaja que presenta el sistema, es el bajo acoplamiento, porque los manejadores solo conocen al kernel, y no hay dependencia entre cada uno de los manejadores. Otra ventaja es la facilidad para manejar otro tipo de dispositivo, API gráfico o demás, ya que solo se deben agregar los manejadores internos y externos correspondientes.

El kernel presenta funciones a bajo y alto nivel, las primitivas que maneja son creadas de acuerdo al hardware, para su sincronización, manejo de procesos y la administración de los recursos.

- Manejadores Internos

Presenta varios manejadores internos, como:

Input Manager – Input Devices: maneja los dispositivos de entrada, HMD (posición), Mouse (digital), Joystick (análogos) y guantes.

Este manejo se realiza de acuerdo a la herencia, ya que posee una clase base y cada una de las clases dependiendo del tipo de dispositivo. La clase base presenta unos métodos principales como el Start, stop, updatedevice.

Para este manejo de los dispositivos se cuenta con un device proxies, donde la aplicación pide datos del dispositivo usando el nombre en la configuración en curso. El manejador de entradas, realiza un lookup de la petición y retorna al proxy el correspondiente dispositivo físico.

Para poder realizar la anterior secuencia, se debe contar con un device store, que permite tener todos los drivers separados de la librería principal y dinámicamente carga los drivers necesarios, todo esto en tiempo de ejecución, presentandose como un Patrón Factory .

- Manejador del Ambiente

Este manejador se encarga de conectar la aplicación, este proceso se lleva a cabo por medio del vjControl, de acuerdo a las necesidades de la aplicación. Este manejador se asemeja a una GUI, donde se puede tener el control dinámicamente de cada uno de los aspectos de la plataforma virtual que se esta ejecutando.

- Manejador de la pantalla

Este manejador encapsula toda la información acerca de las propiedades para la pantalla o ventana de la aplicación, como puede ser el tamaño, la localización, las gráficas (pipeline) y parámetros de vista.

- Manejador Externo

Este presenta el manejador gráfico, que presenta una interfaz para el manejo de la librería grafica deseada, permitiendo un perfecto funcionamiento del entorno gráfico, tal como es OpenGL.

En el momento VR Juggler no presenta más manejadores dentro del manejador externo, pero en un futuro se esta pensando en expandir, por ejemplo el manejador de sonido.

- Aplicaciones Objeto

Las aplicaciones objeto, son las aplicaciones implementadas, el kernel lo reconoce y lo controla, de tal forma que crea el entorno virtual en el que se va a ejecutar, ya que estas aplicaciones no tienen una función principal, sino que son ejecutadas por el kernel disponiendo este de los recursos necesarios para su correcto funcionamiento.

- Loop del kernel

Las partes más importantes dentro de VR juggler son la creación del Input Manager por parte del Kernel, luego la creación de VjGLApp (permite la iniciación de la aplicación gráfica), después se inicializa el VjGLDrawManager de acuerdo a las características transmitidas por el kernel y se empieza a dibujar. Después este VjGLDrawManager crea el VjGLPipe que se encarga del renderizado.

Dentro de este ciclo, se puede observar la dependencia de los cambios existentes en la posición y los procesos de renderizado y rasterizado, donde el encargado del primero proceso es el VjGLPipe y del segundo es el DrawManager, de esta manera se explica la forma en que el VR Juggler realiza todo el proceso de realidad virtual.

3.6. Mecanismos de comunicación con dispositivos de entrada físicos

El manejo de dispositivos de entrada es bastante complejo, puesto que no existe una librería que administre diferentes tipos de periféricos, por lo tanto la única manera de realizar mundos de realidad virtual inmersiva es implementar los drivers y los demás elementos que permiten la interacción con los dispositivos, como manejadores de eventos, hilos de procesamiento, etc. Hoy en día los fabricantes siguen desarrollando los drivers en C++ y por lo tanto es complicado utilizarlos en Java puesto que se hace necesario utilizar mecanismos como JNI (explicado mas adelante) para realizar llamados nativos a los drivers. Actualmente, no existen muchas implementaciones Java de los dispositivos de entrada que son comúnmente utilizados en computación gráfica, lo que genera un obstáculo grande para la creación de mundos virtuales inmersivos en Java.

El lenguaje de programación JAVA no posee una forma directa para reconocer dispositivos de entrada y salida, puesto que es un lenguaje multiplataforma y es el sistema operativo quien debe administrarlos por lo tanto Java no puede involucrarse directamente con el reconocimiento y administración de los periféricos, pero existen algunas soluciones que permiten realizar dichas tareas, una de ellas es la utilización de código nativo (generalmente en **C++**) mediante la interfaz **JNI (Java Native Interface)** para establecer la comunicación directa con cualquier tipo de dispositivo. Otro mecanismo importante es la utilización de **Java Communications API (JC)**, este API permite el manejo de puertos de comunicación seriales y paralelos, por lo tanto es posible realizar los drivers de

cualquier periférico conectado a dichos puertos, siempre y cuando se conozca el protocolo y los parámetros de comunicación.

Existen también periféricos que se comunican por el puerto USB, para acceder a estos dispositivos existen librerías como **SDL** y **JUSB.** A continuación se explicará con más detalle cada uno de estos mecanismos de comunicación.

3.6.1. JNI (Java Native Interface)

Básicamente permite la utilización de código escrito en otro lenguaje de programación (diferente a Java) y actualmente la mayoría de los dispositivos de realidad virtual trabajan con librerías y drivers hechos en C++ lo cual permitiría reutilizar dichas fuentes y utilizarlos en aplicaciones Java. El procedimiento iniciaría básicamente con el cargue del driver en memoria desde la aplicación Java y posteriormente se realizarían los llamados a los procedimientos nativos, los cuales se encargarían del manejo y configuración de los dispositivos.

La principal ventaja de esta solución es la existencia de algunas librerías y drivers para los periféricos más populares lo cual ahorraría tiempo y diseño para establecer funciones de configuración y manejo, pero presenta algunas desventajas en cuanto a mantenimiento puesto que si se desea realizar algún cambio o ajuste es necesario tener el código fuente, editarlo y generar nuevamente una librería que por lo general es un archivo DLL (Dinamic Link Library) **[2].**

3.6.2. Java Communications API (JC)

Este API (Application Programming Interface) permite enumerar los diferentes puertos de comunicaciones que se tengan habilitados en una máquina, además de presentar una serie de clases que permiten realizar el proceso de comunicación con lo periféricos que se encuentren conectados, es decir, permite la programación a bajo nivel de la comunicación con algún tipo de periférico.

Esta librería para la comunicación con los puertos presenta algunas elementos interesantes, puesto que no solo tiene la opción de abrir el puerto y establecer la comunicación directamente, sino además presenta un modelo de eventos que facilitan el trabajo del programador pues le notifican cuando hay información en el puerto y no es necesario estar monitoreando el flujo de información continuamente. También, permite un nivel un poco más alto de portabilidad de la aplicación, frente a **JNI**, brindando un valor agregado con respecto a la flexibilidad que pueden llegar a tener las aplicaciones frente a diferentes tipos de periféricos **[1]**.

3.6.3. SDL [30]

Básicamente SDL es una librería para el desarrollo de aplicaciones multimediales similar a DirectX pero con una gran diferencia, puesto que SDL es multiplataforma y cubre la gran mayoría de sistemas operativos como Windows, Linux, Unix, BeOS, MacOS, etc. Posee un gran acoplamiento con OpenGL y es bastante fácil de usar, además existen implementaciones en la gran mayoría de lenguajes de programación, como C++, ADA, Fortran, Java, Pascal, etc.

SDL permite realizar tareas de desarrollo multimedial como manejo de video, audio, hilos, temporizadores y lo más importante, dispositivos de entrada como CD-ROM y Joysticks, GamePads, estos últimos dispositivos pueden trabajar con conectores USB.

3.6.4. JUSB [31]

Es una librería que permite listar todos los dispositivos de entrada conectados mediante los puertos USB, pero aun se encuentra bajo desarrollo y carece de cierta funcionalidad importante que no la hace muy útil en este momento, puesto que no se puede escribir información en algunos modos de comunicación (ver protocolo USB anexo A).

4. DESCRIPCIÓN DEL PROYECTO

Desde hace mucho tiempo el desarrollo de la computación gráfica y de la realidad virtual se ha visto ligado directamente al lenguaje en el cual se realizan las aplicaciones de dichas áreas (que por lo general es C++), por lo tanto las arquitecturas y los dispositivos de interacción e inmersión están ligados directamente al lenguaje, lo cual hace que lenguajes modernos como Java queden rezagados en estas áreas de la computación.

Actualmente existen algunas soluciones que permiten la comunicación entre dispositivos de entrada y aplicaciones desarrolladas en JAVA, pero dichas soluciones se limitan al manejo de drivers nativos para dispositivos en especial, es decir, no existe una librería de clases Java (API) que permita de manera genérica administrar dispositivos físicos de entrada que facilite el trabajo del desarrollador de aplicaciones de computación grafica.

Viendo el anterior problema, el objetivo general de este proyecto se centra en realizar una librería Java que permita manejar y configurar dispositivos de entrada que permiten la inmersión e interacción con aplicaciones gráficas de realidad virtual. Para cumplir este objetivo se especificaron los siguientes objetivos específicos:

- Recopilar y analizar las especificaciones de los dispositivos de realidad virtual para establecer los requerimientos de la librería a desarrollar.
- Elaborar el análisis de los requerimientos y el diseño de la librería a implementar.
- Desarrollar el plan de pruebas que permita validar el software desarrollado.
- Realizar la implementación de la librería.
- Elaborar la documentación del software y del prototipo empleado en el plan de pruebas.
- Aplicar el plan de pruebas y analizar los resultados obtenidos.

Básicamente el proyecto consiste en el desarrollo de un software (librerías, drivers) para JAVA que permita la utilización de los periféricos de entrada / salida que se utilizan para interactuar con ambientes virtuales y aplicaciones 3D. Tales periféricos son básicamente los joysticks, el guante de datos y el casco, las especificaciones de estos dispositivos estarán sujetas a los equipos de realidad virtual que posee la universidad.

Teniendo en cuenta que las características y especificaciones de los dispositivos que intervienen en este proyecto son bastante particulares entre sí, se determinó que el proceso de desarrollo de la librería se realizaría de manera independiente

para los diferentes tipos de periféricos, manejando una metodología de construcción de software de ciclo de vida en espiral (cíclica) para cada uno.

Debido a las limitaciones que presenta el lenguaje de programación Java para el acceso a dispositivos físicos de entrada, existe un punto importante dentro del proyecto el cual radica en el manejo de drivers, debido a esto es necesario conocer las limitaciones de los mismos al administrar los dispositivos para poder determinar el horizonte de la librería en general, por lo tanto dentro del proceso de construcción de la librería, se encuentra una etapa preliminar en la que se realizarán los drivers para los periféricos con los que cuenta la universidad.

Con base en lo anterior el proyecto está constituido de la siguiente manera:

- **Primera Fase: Construcción de la librería para el Guante de Datos**

  A continuación se describen las principales actividades que componen esta fase.
  - Estudio de especificaciones: en esta actividad se recopila la información concerniente a las características físicas del dispositivo, en especial del periférico que posee la universidad. En esta etapa es necesaria la información acerca de los protocolos de comunicación, parámetros de configuración, estructura de los paquetes de datos, etc.
  - Definición del mecanismo de comunicación: en esta etapa se estudian las diferentes opciones que permite el lenguaje Java para establecer la comunicación con el periférico, esto orientado directamente hacia la construcción del driver y posteriormente se define la mejor alternativa.
  - Construcción del driver: Una vez definido el mecanismo de comunicación se construye en esta etapa el driver correspondiente, utilizando la información del protocolo del dispositivo y los parámetros de inicialización obtenidos en la actividad de estudio de especificaciones.
  - Pruebas del driver para el guante de datos: debido al ancho de banda que maneja este dispositivo (ver marco teórico), es fundamental que el driver obtenga la mayor cantidad de información del periférico, esto para garantizar un correcto funcionamiento al momento de interactuar en un mundo virtual, por lo tanto se realizaran pruebas de desempeño del driver.
  - Definición de requerimientos de la librería: en esta fase se especifican de acuerdo a las características del dispositivo y del driver los requerimientos que debe suplir la librería para el dispositivo en especial.
  - Diseño de la librería: en esta fase se realiza el diseño de la librería, basado en los requerimientos definidos en la etapa anterior.

- Implementación de la librería: con base en el diseño se construirá el código fuente de la librería para el tipo de dispositivo en desarrollo.
- Pruebas de implementación: en esta fase se realizarán pruebas que permitan validar la implementación realizada en la etapa anterior.
- Definición de requerimientos de la aplicación de configuración: en esta etapa se definirán las características que debe tener la aplicación de configuración del dispositivo y la funcionalidad que debe suplir la librería para dicha configuración a nivel general.
- Diseño e implementación de la aplicación de configuración: en esta etapa se estudiarán los requerimientos para realizar el diseño y la implementación de la aplicación de configuración.
- Pruebas: en esta etapa se validará el correcto funcionamiento de la aplicación de configuración y la librería.

- **Segunda fase: Construcción de la librería para el Casco Virtual**

  - Estudio de especificaciones: en esta actividad se recopila la información concerniente a las características físicas del casco virtual. En esta etapa es necesaria la información acerca de los protocolos de comunicación, parámetros de configuración, estructura de los paquetes de datos, etc.
  - Análisis del driver nativo del fabricante: para recopilar información extra del casco virtual que posee la universidad, se hace necesario estudiar el driver que provee el fabricante y el software de desarrollo y configuración.
  - Definición del mecanismo de comunicación: de acuerdo a la información obtenida en las dos primeras actividades se define la mejor alternativa para construir el driver de comunicación con el periférico.
  - Construcción del driver: En esta etapa se debe utilizar el mecanismo de definido en la actividad previa para aplicar el protocolo de comunicación con los parámetros correspondientes.
  - Pruebas del driver: en esta fase se realizarán pruebas que permitan validar la comunicación con el dispositivo físico.
  - Definición de requerimientos de la librería: en esta fase se especifican de acuerdo a las características del dispositivo y del driver los requerimientos que debe suplir la librería para el dispositivo.
  - Diseño de la librería: en esta fase se realiza el diseño de la librería, basado en los requerimientos definidos en la etapa anterior.
  - Implementación de la librería: con base en el diseño se construirá el código fuente de la librería para el casco virtual.
  - Definición de requerimientos de la aplicación de configuración: en esta etapa se definirán las características que debe tener la

aplicación de configuración del dispositivo y la funcionalidad que debe suplir la librería.
    - Diseño e implementación de la aplicación de configuración: en esta etapa se estudiarán los requerimientos para realizar el diseño y la implementación de la aplicación de configuración.
    - Pruebas: en esta etapa se validará el correcto funcionamiento de la librería y la aplicación de configuración.

- **Tercera Fase: Construcción de la librería para los joysticks**

    - Estudio de especificaciones: Al igual que en las fases anteriores se hace necesario una búsqueda de información de las características y especificaciones de los joysticks existentes y en particular de los periféricos que posee la universidad.
    - Definición del mecanismo de comunicación: en esta etapa se evaluarán los pros y contras de los diferentes mecanismos de comunicación USB en Java y se definirá cual será el utilizado por la librería.
    - Definición de requerimientos: con base en la información obtenida en la primera fase y en la funcionalidad proporcionada por el mecanismo de comunicación seleccionado se definirán los requerimientos que deberá satisfacer la librería para este tipo de dispositivos.
    - Diseño de la librería para los joysticks: se realizará el diseño de esta parte de la librería de acuerdo con la definición de requerimientos de la fase previa.
    - Implementación de la librería: se construirá el código fuente de acuerdo con el diseño previo.
    - Pruebas: se realizarán pruebas de implementación para validar coherencias, estructura del código fuente y funcionalidad.

Nota: No se realizará aplicación de configuración de estos dispositivos, pues dicha configuración se realiza a nivel de sistema operativo, con excepción del manejo del feedback que poseen algunos joysticks, pero este no será tomado en cuenta.

- **Cuarta fase: Diseño general de la librería**

    - Estudio de patrones de diseño: se evaluarán los diferentes patrones de diseño que permiten integrar de ser posible las librerías desarrolladas en las fases previas.
    - Definición de patrones de diseño: se seleccionarán las mejores alternativas en cuanto al diseño global de la librería, para poder integrar los módulos desarrollados.

- Estudio de mecanismos de comunicación con dispositivos de entrada lógicos: se realizará un estudio que permita definir las alternativas de comunicación entre los dispositivos y las aplicaciones de computación gráfica que desarrollará el usuario final.
- Definición de mecanismos de comunicación con dispositivos de entrada lógicos: se evaluarán las alternativas definidas en la fase anterior para seleccionar la mejor solución al problema de comunicación entre aplicación y dispositivos.

- **Quinta fase: Pruebas y Análisis**

  - Ejecución del plan de pruebas: se ejecutará el plan que se definió dentro de las actividades previas a la cuarta fase del proyecto.
  - Análisis de resultados del plan de pruebas: una vez se haya ejecutado el plan de pruebas se evaluarán las variables medidas durante la ejecución del plan y se documentarán los resultados del mismo.

- **Sexta fase: Conclusiones y Recomendaciones**

  En esta fase se determinarán los principales aportes, problemas, ventajas y desventajas de la solución desarrollada; posteriormente con base en dicho análisis se realizarán las recomendaciones que sugerirán los investigadores orientadas, hacia futuros proyectos.

5. DESARROLLO DEL PROYECTO

Para el desarrollo del proyecto se empleó una metodología de investigación y una de ingeniería de software para la construcción del API que extiende la funcionalidad del lenguaje de programación JAVA.

La metodología de investigación se basa en primera instancia en una fase exploratoria, que define las especificaciones de los dispositivos, las opciones dentro de la comunicación con los mismos y además las ventajas existentes dentro de la realidad virtual. Posteriormente se realizó una fase de explicación y análisis de los resultados obtenidos.

El modelo para la construcción del software es en espiral, por la facilidad para validar cada uno de los niveles de investigación y construcción realizados, dentro de la espiral planteada, se tiene unos niveles o ciclos, que se cumplen con la construcción de cada uno de los dispositivos.

Por ejemplo, para el guante, se puede observar las fases dentro de la construcción de la librería como son:

- el estudio preliminar
- la construcción del driver, con su correspondiente validación
- requerimientos, diseño e implementación de la librería correspondiente y su validación

Después de realizar estas etapas con cada uno de los dispositivos, se diseña la librería en general y se prueba con la aplicación o mundo virtual creado.

5.1. Guante de Datos

A continuación se describirá el proceso de la investigación que se llevo a cabo para la realización de la librería que permite realizar el manejo y configuración de los guantes de datos de realidad virtual.

5.1.1. Estudios Preliminares

En este apartado se especifican los estudios realizados y los mecanismos de comunicación con el dispositivo.

5.1.1.1. Estudio de especificaciones

Al comienzo se estudiaron las características principales de algunos guantes, donde se pueden apreciar los aspectos semejantes con el existente en la

Pontificia Universidad Javeriana, como por ejemplo, los grados de libertad, la velocidad de conexión y el número de sensores.

Los guantes en su mayoría están confeccionados de lycra, con pequeñas fibras que transmiten los valores de los sensores, para poder retransmitirlos al computador por medio del puerto serial.

Los guantes de datos se comunican por el puerto serial en donde presentan diferentes velocidades como se observa en la tabla 3 de comparación de cada uno de los guantes investigados.

GUANTES	SENSORES GRADOS DE LIBERTAD	PUERTO	CAPTURAS	TASA O RATA	VALOR
P5 GLOVE	6 (x,y,z, yaw, pitch, roll)	USB	5 medidas 0 - 90°	60 Hz	59
5DT GLOVE 5	1 sensor por dedo, y 2 para pitch roll	Serial, USB	256 posiciones	75 Hz o 9600 b	895 c/u
5DT GLOVE 14	2 sensores por dedo	Serial, USB	256 posiciones	200 Hz	
5DT GLOVE MRI	2 o 1 por dedo, y 2 pitch roll	Serial, USB	256 posiciones	19,2 Kb o 200Hz	
DG5-VHAND	5 sensores	Serial	1024 posiciones	20 MHz	485 c/u
X-IST GLOVE	5 sensores	Serial, USB		100-200Hz	1279 c/u
PINCH GLOVE	1 por dedo y 6 grados de libertad	Serial		19,2 Kb	1899
CYBER GLOVE	18 o 22 sensores	Serial		50-60 Hz o 115,2 Kb	

Tabla 3. Comparación entre guantes de realidad virtual

Dentro de la misma marca del guante existente en la universidad (5DT), se puede apreciar que existen guantes con 7, 14 o 16 sensores, aportando una mayor precisión al momento de obtener el movimiento de cada dedo, por presentar varios sensores en cada uno.

El guante a utilizar tiene 7 sensores (5-W) y se comunica a una velocidad de 9600 bps, con 8 bits de datos y uno de parada sin paridad, el protocolo de comunicación y la estructura de los paquetes de datos se explica a continuación**.**

La estructura del paquete de datos se puede ver en la figura 18:

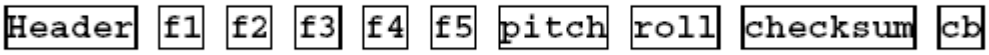


Figura 18 Estructura del paquete de datos

El encabezado (header) siempre es un byte con un valor de 0x80 en hexadecimal (128 entero)

cb determina el tipo de guante conectado, específicamente si el byte es 1 (entero) el guante conectado es el derecho y si el byte es 64(entero) entonces el guante conectado es el izquierdo.

Si el guante conectado es derecho entonces se tiene que:

f1= pulgar, **f2**= índice, **f3**= medio, **f4**= anular, **f5**= meñique

Si el guante conectado es izquierdo se tiene que:

f1= meñique, **f2**= anular, **f3**= medio, **f4**= índice, **f5**= pulgar

Los valores de cada sensor al igual que los de pitch y roll son bytes con valores entre 0 y 255, una vez se establece la comunicación con el dispositivo, este empieza a enviar paquetes con la estructura definida previamente.

5.1.1.2. Definición del mecanismo de comunicación

Para el mecanismo de comunicación por el puerto serial, se buscaron diferentes opciones que presenta el lenguaje JAVA, entre las más importantes se encuentra Java Communications API (javacomm), esta librería presenta algunos problemas con la pérdida de información, pero sin un nivel de importancia significante para la escogencia del mismo, ya que se probó el API en el modo manual y no se presentó ningún problema relacionado con la pérdida de información.

5.1.2. Proceso de creación del driver

A continuación se describe los procesos de construcción y las respectivas pruebas del driver del guante de datos.

5.1.2.1. Construcción del driver

De acuerdo a la información que se tiene de la transmisión que realiza el guante, se observan los datos en cada una de las posiciones dentro de la trama, en donde dependiendo de la mano, se puede reconocer el dedo en movimiento o el movimiento realizado por la mano.

Antes de empezar a capturar los valores para cada uno de los sensores dentro de la trama se corrigió la captura de la trama, es decir, se validó la trama de acuerdo

al bit que determina el tipo de guante conectado y el bit de parada, dando como resultado la pérdida de algunas tramas dentro de la transmisión.

Partiendo de lo anterior, el driver se encarga de obtener los datos de los sensores, en donde cada uno tiene un valor entre 0 y 255. El bit de reconocimiento del guante (izquierdo ó derecho), puede tener un valor de 1 ó 64, en donde 1 es el derecho y 64 el izquierdo. Los valores de pitch y roll, tienen un rango igual al de la información de cada dedo, de acuerdo a la posición de la mano.

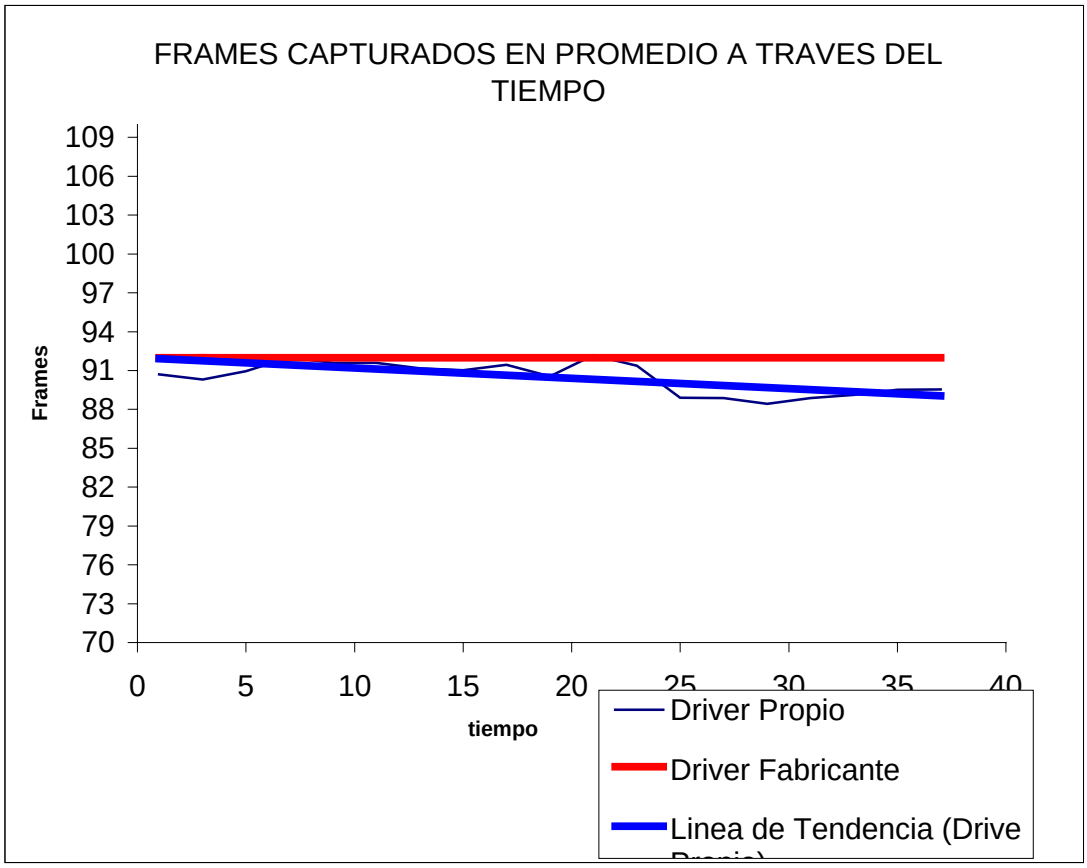
5.1.2.2. Pruebas del driver

Las pruebas para esta fase, se centran en el número de tramas capturadas, para establecer las tramas perdidas de acuerdo a la velocidad de transmisión, de esta manera poder establecer si el mecanismo de comunicación utilizado y la corrección de la captura de tramas es óptimo.

Esta pruebas se dividieron en dos, la primera trata de validar la comunicación con el dispositivo, mientras que la segunda valida y mide el número de datos capturados en un determinado tiempo de acuerdo a los intervalos de tiempo especificados y el tamaño de la muestra.

A parte de las pruebas descritas anteriormente, se realizaron unas, que pretendían reconocer si el driver funcionaba, bajo otro sistema operativo, como fue Solaris, con el Java Communication API para esta plataforma.

Para estas pruebas, la muestra es el número de tramas capturadas en un determinado tiempo, en la tabla 4 se muestran los valores promedio de captura de paquetes de datos en un determinado tiempo.



Segundos	1	3	5	7	9	11	13	15	17	19
Media	92,25	91,87	91,10	90,29	89,52	89,74	89,68	90,18	88,73	88,81
Desviación	3,06	3,57	3,21	2,22	2,85	2,09	2,16	1,43	1,39	1,43
Tramas perdidas	1,75	2,13	2,90	3,71	4,48	4,26	4,32	3,82	5,27	5,19
Porcentaje %	1,86	2,27	3,08	3,94	4,76	4,53	4,6	4,06	5,6	5,52

Tabla 4 Valores estadísticos de la prueba del driver del guante de datos

Teniendo en cuenta que el dispositivo envía paquetes de datos a una razón de 94 tramas por segundo, se tiene que la pérdida de información está dentro del 1% y el 5%, obteniendo en promedio 89 paquetes por segundo, un paquete menos que el número de paquetes de datos que maneja el driver del fabricante, por lo tanto se determinó que el driver desarrollado era adecuado.

Estas pruebas preliminares tenían como objetivo establecer la validez del mecanismo de comunicación con el guante, obteniendo los datos descritos en la tabla 4, en donde las tramas perdidas no exceden de 5.

Después de realizar estas pruebas se tomaron muestras de 20 repeticiones cada una, desde 1 hasta 39 segundos, es decir, que la primera muestra fue de 20 repeticiones de un segundo cada uno.

Los datos de esta prueba pueden ser vistos en el anexo C en donde se pueden apreciar todos los datos capturados.

Al finalizar estas pruebas se pudo concluir que el número de tramas perdidas no es significante para el tipo de aplicaciones en la que se va a utilizar, ya que las perdidas tampoco sobrepasan el 5% de las tramas totales por segundo.

La desviación que se tiene esta dentro de un rango de 0 a 3, de acuerdo a estos datos se confirma que los datos perdidos en la validación de la trama por parte del driver y la pérdida que puede presentar Java Communications API, no es considerable.

La prueba para el funcionamiento del driver bajo el sistema operativo Solaris, después de la instalación de javacomm para esta plataforma se obtuvo un comportamiento similar al de las pruebas realizadas en sistemas operativos Windows, con el valor agregado de no utilizar el driver creado por los fabricantes que tiene un costo elevado en el mercado (alrededor de 500 dólares).

5.1.3. Proceso de creación de la librería

Para la creación de la librería, se siguieron los pasos de la definición de requerimientos, el diseño y sus correspondientes pruebas para la validación de la misma

5.1.3.1. Definición de requerimientos

La definición de los requerimientos esta ligada a los casos de uso presentados en la figura 19 donde se observan los siguientes casos de uso, con su documentación en el anexo D1:

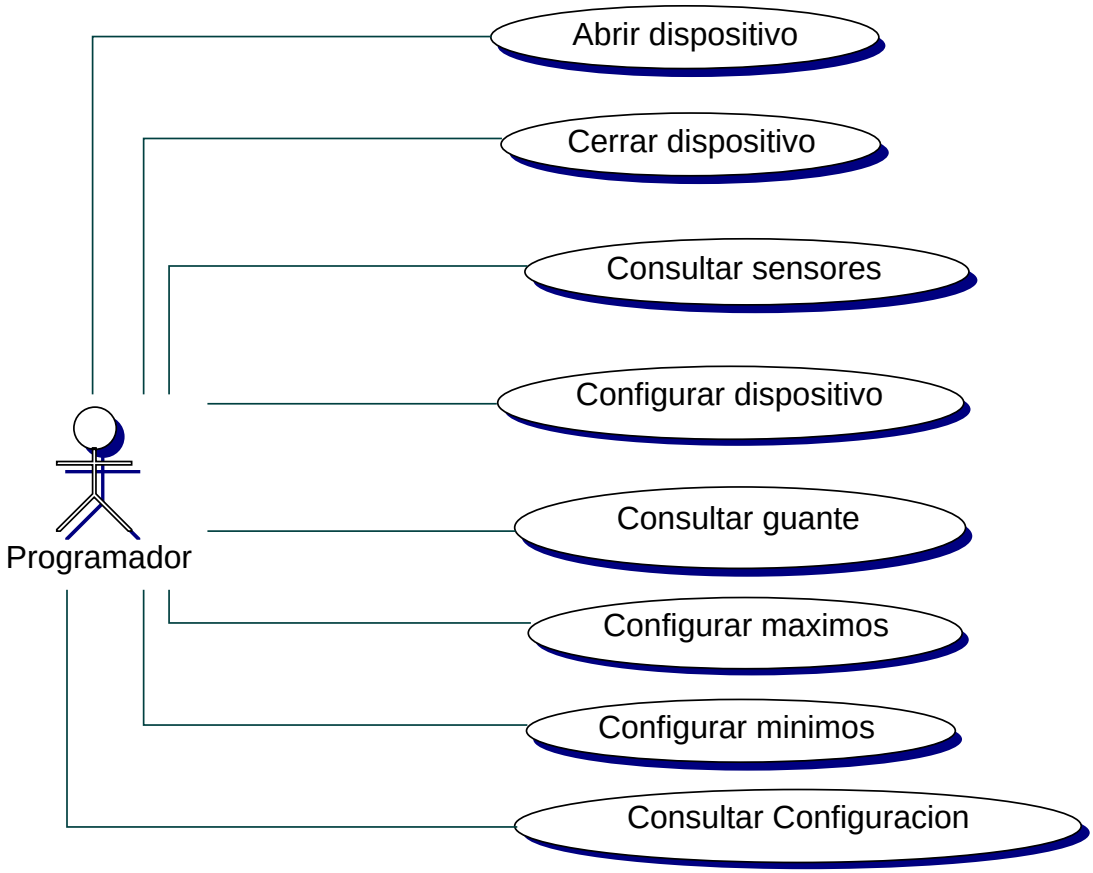


Figura 19 Casos de uso de la librería del guante de datos

5.1.3.2. Diseño

El diseño de la librería basado en los requerimientos descritos se puede observar en la figura 20:

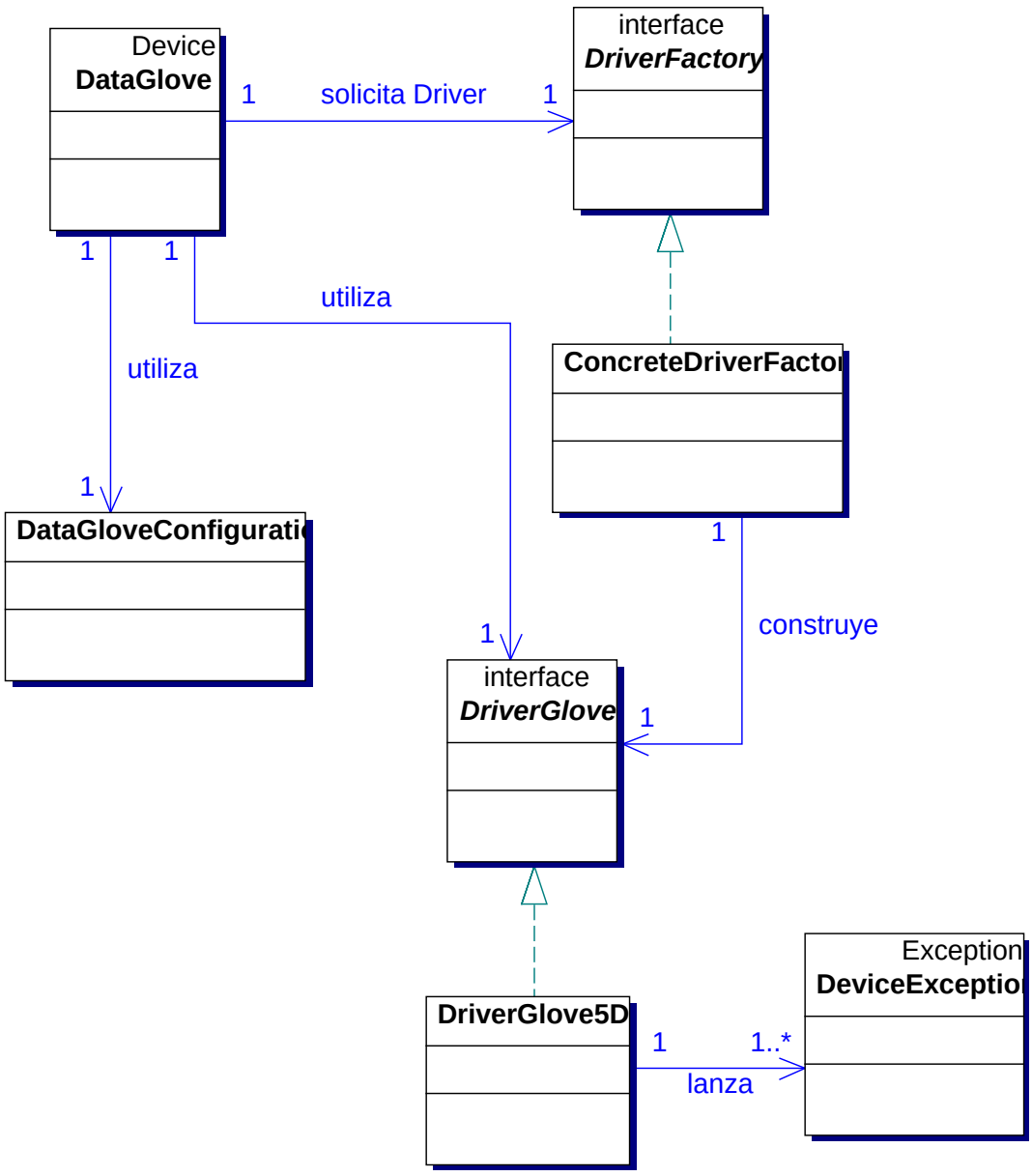


Figura 20 Diagrama de clases de la librería del guante

A continuación se presentará una breve descripción de cada una de las clases que conforman la librería que administrará los guantes de datos, el diagrama de clases a nivel de implementación se muestra en el anexo B3.

- **DriverGlove (interface):** en donde se encuentran los métodos que se van a implementar, como abrir dispositivo, cerrar dispositivo, consultar sensores, consultar guante, etc.

- **DriverGlove5DT:** aquí viene la implementación de los métodos de la interface específicos para el dispositivo del guante 5DT.
- **DeviceException:** esta clase maneja los errores que se puede presentar al momento de utilizar el driver, ya sea abriendo, cerrando el dispositivo o en cualquiera de sus métodos.
- **ConcreteDriverFactory:** esta clase va a cargar el driver especifico del dispositivo solicitado.
- **DriverFactory (interfaz):** donde se encuentran los métodos de creación de los driver del guante. createDriverGlove().
- **DataGlove:** esta clase extiende de la interface device, en donde se encuentran los datos o atributos propios del guante.
- **Device (interfaz):** La clase genérica de los dispositivos de realidad virtual.
- **DataGloveConfiguration:** esta clase es la que permite guardar y/o consultar los datos del archivo de configuración del guante.

5.1.3.3. Pruebas de implementación

Se realizaron varias pruebas para reconocer los datos que se estaban capturando, estos datos se estaban mostrando por pantalla para observar el comportamiento de la librería ver figuras 21 a la 24, a veces se observa un problema con el flujo de datos al inicio del programa, este error fue corregido sincronizando los paquetes de datos, esto para evitar desfases en la lectura de la información.

Las figuras muestran los datos de las tramas capturadas de acuerdo a 2 diferentes posiciones, las figuras 21 y 22, muestran el guante en una posición sobre la mesa, donde nadie esta interactuando con el dispositivo, se observa que los valores son iguales en los dedos, mientras que para el pitch y roll, hay una pequeña variación, esto debido a que los datos siempre están oscilando.

Los valores obtenidos están dentro de un rango permitido, ya que es de más o menos 1, entre 255 posiciones que puede tomar la posición de pitch and roll, donde 1 representa más o menos el 0.4% de los datos. Este porcentaje expresa un valor mínimo dentro de los cambios, que pueden verse afectados por movimientos del mismo actor al utilizarlos, por vibraciones en la extremidad de las manos, lecturas enviadas por los sensores descalibrados, etc.

```
128 0 0 0 0 0 142 125 243 1
128 0 0 0 0 0 142 125 243 1
128 0 0 0 0 0 142 125 243 1
128 0 0 0 0 0 142 125 243 1
128 0 0 0 0 0 142 125 243 1
```

Figura 21 Paquetes de datos obtenidos mediante el GloveManager en la Prueba 1

```
128 0 0 0 0 0 142 124 242 1
128 0 0 0 0 0 142 124 242 1
128 0 0 0 0 0 142 124 242 1
128 0 0 0 0 0 142 124 242 1
128 0 0 0 0 0 142 124 242 1
```

Figura 22 Paquetes de datos obtenidos mediante la aplicación Java en la Prueba 1

En las figuras 23 y 24, se muestran los valores del guante, que utilizaba el actor, que pretendía tener siempre la misma postura, para la medición de esta prueba.

Aquí se observa una variación en los datos de los dedos, que se puede atribuir a un movimiento involuntario del actor, siendo insignificante para la posición o posible relación que pueda llegar a hacer el programador con estos datos, esta variación es de 0.4 %, bastante pequeño para poder descartar este valor, ya que no se ha realizado un verdadero cambio para que se vea reflejado en un futuro en una aplicación.

```
128 100 255 132 255 255 143 106 250 1
128 100 255 132 255 255 143 106 250 1
128 100 255 132 255 255 143 106 250 1
128 100 255 132 255 255 143 106 250 1
128 100 255 132 255 255 143 106 250 1
```

Figura 23 Paquetes de datos obtenidos mediante el GloveManager en la Prueba 2

```
128 101 255 133 255 255 143 106 250 1
128 101 255 133 255 255 143 106 250 1
128 101 255 133 255 255 143 106 250 1
128 101 255 133 255 255 143 106 250 1
128 101 255 133 255 255 143 105 249 1
```

Figura 24 Paquetes de datos obtenidos mediante la aplicación Java en la Prueba 2

5.1.4. Procesos para la creación de la aplicación de configuración

A continuación se muestran los pasos para la creación de la aplicación de configuración, como es la definición de los requerimientos, el diseño e implementación y las pruebas para validarla.

5.1.4.1. Definición de requerimientos

Los requerimientos de la aplicación de configuración se determinaron a partir de la aplicación de configuración que provee el fabricante, la interfaz de la aplicación se muestra en la figura 25.

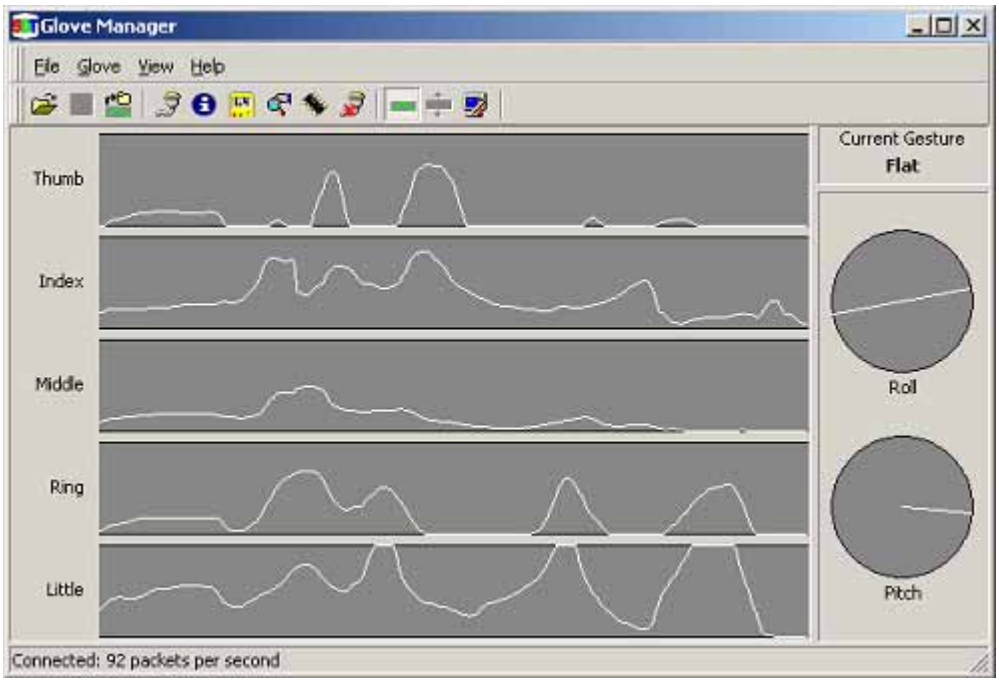


Figura 25 Administrador del guante de datos 5DT suministrado por el fabricante

En esta aplicación cada sensor es representado por una línea continua en la cual varia la amplitud de manera análoga al comportamiento de un movimiento armónico simple.

A partir de esta aplicación se determinó que era necesario mostrar de manera gráfica el estado y variación actual de cada uno de los sensores, para poder configurar rangos máximos y mínimos de una manera sencilla.

5.1.4.2. Diseño e implementación

Con base en la aplicación que suministra el fabricante se diseño e implementó una aplicación de configuración cuya interfaz se muestra en la figura 26.

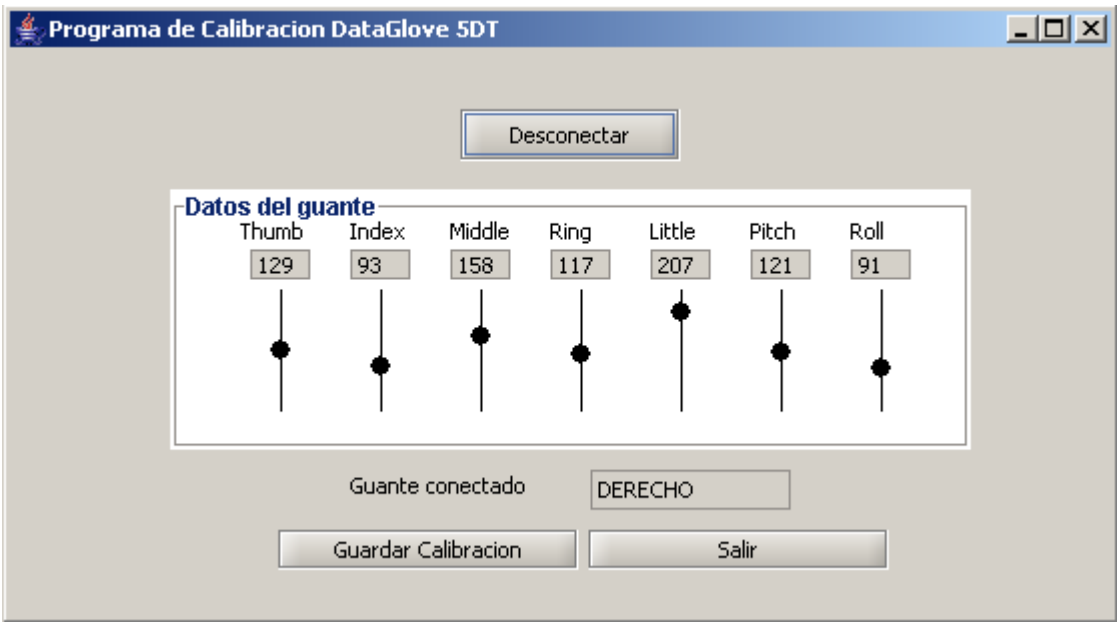


Figura 26 Administrador del guante de datos 5DT desarrollado en Java

En esta aplicación cada sensor es representado por una línea en la cual se encuentra un indicador, el cual varía su posición de acuerdo a los movimientos del dispositivo físico.

5.1.4.3. Pruebas

Las pruebas de la librería se basaron en la muestra de los datos recogidos, estos datos son mostrados a través del programa de configuración implementado, este programa, permitió reconocer fallas existentes en la construcción de la librería como el reconocimiento del dispositivo cuando no esta conectado al puerto serial, este problema fue resuelto gracias al método ***ready***() de la clase ***BufferedReader***.

Los diferentes datos son mostrados en los cuadros de texto, estos valores están dentro del rango de 0 a 255, dependiendo del dedo que se esté flexionando o moviendo, además muestra que guante esta conectado (derecho e izquierdo), como se puede observar en la figura 26.

5.1.5. Conclusiones Parciales

- Los mecanismos para la comunicación con el dispositivo empleados en el proceso de la construcción del driver, presentan un rendimiento óptimo al momento de capturar la información, ya que los datos que se pueden perder son despreciables, por la tasa de transmisión del dispositivo.

- La librería permite la utilización del driver de manera transparente para el usuario y facilita la expansión de la misma, debido a que se puede integrar drivers de diferentes guantes sin tener que cambiar su implementación.
- La aplicación de configuración, permite guardar los datos necesarios y solicitados por el usuario, mejorando la abstracción de los mismos en aplicaciones en Java de realidad virtual.

5.2. Cascos para realidad virtual

A continuación se describirá el proceso que se llevó a cabo para la realización de la librería, que permite el manejo y configuración de los cascos de realidad virtual.

5.2.1. Estudios Preliminares

Para poder realizar la construcción del driver, se estudiaron las características propias del casco VFX3D, tales como su protocolo y sus parámetros de comunicación, además de los mecanismos para establecer la comunicación con el dispositivo.

5.2.1.1. Estudio de especificaciones[35]

Dentro del ámbito actual existe un gran número de dispositivos de inmersión que permiten al usuario observar mundos virtuales, pero muchos de ellos no cuentan con sensores de movimiento que hagan posible la interacción con el mundo, entre estos dispositivos se cuentan los binoculares 3D, las gafas de realidad virtual y algunos cascos (Head Mounted Display) que no permiten dicha interacción. Por lo tanto, solamente se tendrán en cuenta únicamente aquellos dispositivos que sean entrada / salida.

Los flujos de salida son generalmente manejados por la tarjeta de video directamente, por lo tanto lo más importante a estudiar son los flujos de entrada de información, la cual representa el estado actual de sensores de movimiento. La Pontificia Universidad Javeriana cuenta con un casco de realidad virtual VFX3D**[34],** sobre el cual se realizó un gran número de pruebas para determinar el protocolo de comunicación y los parámetros de configuración y la estructura de los paquetes de datos, en la siguiente sección se describirá el proceso que se realizó para determinar esta información. Como información inicial del dispositivo se conocía únicamente que estaba conformado por los siguientes tres sensores de movimiento:

- Pitch: Este sensor determina la posición en grados del movimiento de la cabeza en sentido vertical (únicamente), con un rango de 140 grados.

- Roll: Este sensor determina la posición en grados del movimiento que la cabeza realiza cuando se inclina hacia cualquiera de los dos hombros, esto con un rango de 140 grados.
- Yaw: Este sensor determina el movimiento que la cabeza realiza al girar sobre su propio eje, este tiene un rango de 360 grados.

Además se conocía por simple inspección que el casco VFX3D se comunicaba con el computador mediante un cable serial pero se desconocía todo lo referente a la comunicación, a continuación se describirá el proceso que se desarrolló para determinar dicha información.

5.2.1.2. Investigación de especificaciones del casco VFX3D

Dentro de la comunicación por puerto serial es indispensable conocer el protocolo de comunicación, los parámetros de velocidad, bits de parada, paridad y el número de datos que manejan los paquetes de datos. Para obtener dicha información inicialmente se estudiaron las siguientes alternativas:

- **Análisis del driver nativo**: en primera instancia se estudió la posibilidad de analizar el driver que provee el fabricante para decompilarlo y conocer el manejo interno que se realiza allí, pero esta alternativa no fue realizable pues no se encontraron herramientas óptimas para hacerlo.
- **Ensayo y error:** al no hallar otra alternativa se decidió abrir el puerto serial y observar el comportamiento del dispositivo al enviarle diferentes caracteres y secuencias de datos, pero no fue posible determinar nada, ya que se desconocían los parámetros de configuración y la información que devolvía el casco eran siempre los mismos caracteres y nunca devolvía la información actual de los sensores.
- **Monitoreo del puerto:** en última instancia se empleó un monitor de puertos seriales (**sniffer**), con esta aplicación se determinaron los parámetros de configuración de la comunicación con el casco:

    - Velocidad (Baud Rate): 14.400 bps
    - Bits de Parada: 2 bits
    - Bits de datos: 8
    - Paridad: par

    El sniffer empleado lleva de nombre **Serial Monitor[44],** pero éste no permitía monitorear el flujo información del puerto, por lo tanto se utilizó un sniffer llamado **HHD Serial Monitor[43],** el cual si permitía realizar dicho monitoreo, de esta forma se utilizó una aplicación que hacia uso del driver del fabricante y así se determinó el protocolo de comunicación, el cual se encuentra en el archivo ***ProtocoloVFX3D.doc*** con los archivos fuente de la librería.

5.2.1.3. Definición del mecanismo de comunicación

Una vez se obtuvo la información necesaria del casco de realidad virtual VFX3D se decidió estudiar la posibilidad de emplear Java Communications API (javacomm), para la comunicación por puerto serial, pero la velocidad que maneja el casco no es soportada por javacomm, por lo tanto se decidió realizar un puente en código nativo (C++) y emplear **JNI** para que el driver Java, que se desarrollaría posteriormente, utilizara dicha librería de comunicación.

5.2.1.3.1. Características del puente nativo

Básicamente este puente permite la siguiente funcionalidad:

- Abrir el puerto: se le debe especificar el nombre del puerto que se desea abrir y devuelve el file descriptor del puerto.
- Cerrar el puerto: se le debe pasar como parámetro el file descriptor, para poder cerrar el puerto que especifica dicho descriptor.
- Leer información: permite leer la información entrante por el puerto.
- Escribir información: permite enviar determinada cantidad de datos por el puerto serial.
- Configuración de parámetros: permite configurar parámetros como la velocidad de comunicación.

La documentación correspondiente a este puente nativo se encuentra en el anexo F.

5.2.2. Proceso de creación del driver

Para la creación del driver, se creó un puente nativo, para poder establecer la comunicación, además se estructuró los datos y se mapeó la información para que fuera coherente, de acuerdo a las especificaciones encontradas para entregar valores lógicos a la librería.

5.2.2.1. Construcción del driver

Una vez se construyó el puente nativo se iniciaron las labores de construcción del driver, en donde se emplea una clase auxiliar que permite el manejo nativo mediante JNI; la clase principal del driver tiene básicamente la misma funcionalidad del puente nativo, pero en el driver se encuentra todo el manejo del protocolo de comunicación y de los descriptores, dejando al puente todo el manejo de la comunicación, logrando una aplicación transparente. La documentación del driver se encuentra en el anexo F con la documentación del puente nativo.

El driver también se encarga de realizar el mapeo de los datos físicos que se reciben desde el dispositivo a los valores lógicos que serán entregados a la librería y por tanto al usuario final, a continuación se describirá la estructura del paquete de datos y el proceso de mapeo que realiza el driver.

5.2.2.1.1. Estructura del paquete de datos

Al utilizar el puente nativo se emplea un método que devuelve un arreglo de bytes que corresponden al flujo de entrada por dicho puerto, una vez se ha establecido la comunicación con el casco, el dispositivo empieza a transmitir un paquete de datos de 14 bytes, al principio de la investigación se creía que existían tres bytes que representaban los sensores del dispositivo; la primera implementación del driver buscaba tres bytes dentro del paquete de datos, pero al realizar las pruebas parciales los datos arrojados por la aplicación de prueba no eran congruentes con el estado actual del dispositivo físico.

Se determinó posteriormente que la estructura del paquete de datos no era correcta puesto que los valores que arrojaba el driver del fabricante eran valores hasta de 65536, por lo tanto la información de los sensores debía ser representada por 2 bytes, por la tanto el tamaño del paquete es 7 con valores entre 0 y 65536 como se muestra en la figura 27.

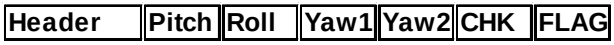


Figura 27 Estructura del paquete de datos del casco VFX3D

El encabezado tiene un valor de 840C en Hexadecimal (33804 decimal), el campo FLAG tiene un valor de 210D Hexadecimal (8461 decimal), se tienen dos valores para representar el sensor Yaw, ya que son necesarios para establecer la posición relativa del casco.

5.2.2.1.2. Proceso de mapeo

Puesto que el puente nativo de acceso al puerto serial devuelve byte a byte la información del flujo de entrada, se convirtieron los paquetes de 14 bytes a paquetes de 7 elementos, para realizar esto se tomaban parejas de bytes y se concatenan utilizando propiedades de suma de potencias, de esta forma fué posible determinar los valores que correspondían a los sensores.

Para los sensores de Pitch y Roll existía un solo valor que representaba el estado actual del sensor, este valor se encontraba en un rango entre 0 y 16380, por lo tanto se realizó una conversión aritmética para establecer dicho rango entre –12743 y +12743 los cuales representan valores en grados entre +70º y –70º.

Para el sensor de Yaw se utilizan dos datos del paquete de información que determinan la posición exacta de rotación de dicho sensor, los valores tiene un rango aproximado entre 5000 y 12000, este rango varía de acuerdo a la posición del sensor *Pitch*, para el mapeo de la información se realizaron operaciones aritméticas que permitieron establecer un rango entre -32768 y +32768 lo cual corresponde en grados a un rango entre –180º y +180º que representan la circunferencia en su totalidad.

4.2.2.2 Pruebas del driver

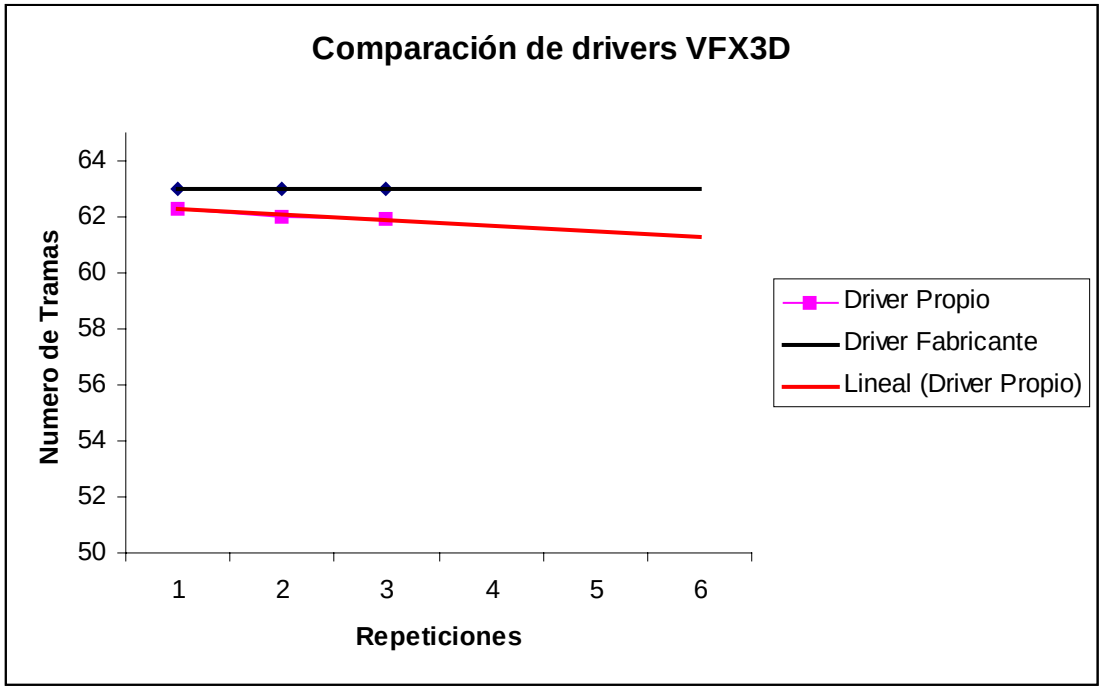
Una vez re construyó el driver para el casco VFX3D se realizaron pruebas que permitieron validar los valores de cada uno de los sensores, para realizar esto se realizaron comparaciones entre el driver desarrollado y el driver del fabricante, se determinaron las diferencias entre los valores de los sensores arrojados por ambos drivers y se promediaron los datos de 10 pruebas diferentes. En la tabla 5 se muestran los resultados de dichas pruebas.

Sensor	**Pitch**	**Roll**	**Yaw**
Diferencia Promedio	424	1706	1166
Error	0,646972656	2,603149414	1,779174805

Tabla 5 Datos de las pruebas del driver VFX3D

De las pruebas se determinó que el error para cada uno de los sensores era despreciable ya que este no supera el 3% y teniendo en cuenta que los rangos de los valores de dichos sensores está entre 0 y 65536 (0 a 360 grados) las posibles diferencias no llegarían a representar ni siquiera un grado de movimiento, por lo tanto se determinó que el driver era adecuado para representar el estado actual del casco VFX3D.

A parte de estas pruebas tambien se compararon los datos del driver del fabricante con respecto al driver creado, la comparacion se baso en el conteo del numero de paquetes que se obtienen en un segundo, para estas pruebas se utilizon el monitor del puerto serial, para un determinado tiempo (30 s), se contaron los paquetes capturados, los resultados se muestran en la tabla 6.



Driver Fabricante	Numero de tramas	Driver Propio	Numero de tramas
	1485,85		1495,35
30	1511,85	30	1487,35
24	1531,85	24	1485,92
Suma	4529,55	Suma	4468,62
Promedio	1509,85	Promedio	1489,54
Paquetes en segundos	62,91041667	Paquetes en segundos	62,06416667

Tabla 6 Comparación entre driver fabricante y propio

Teniendo en cuenta el número de paquetes que se recibe por parte del driver del fabricante y el propio, los valores son muy cercanos, donde la diferencia no es mayor al 2%, que representa un paquete en un segundo, considerando que este valor es despreciable de acuerdo a la velocidad de movimiento y grados de libertad que tiene el dispositivo.

El número de paquetes que se capturan con el driver de fabricante, tiene un promedio de 63, el driver construido tiene un promedio de 62, por lo tanto se determinó que el driver desarrollado es adecuado para su uso dentro de la librería.

5.2.3. Proceso para la creación de la librería

Para la creación de la librería, se definen los requerimientos y el diseño, partiendo de estas definiciones y documentos dejados por la fase anterior, se llega al paso de la implementación de la librería, con su etapa de pruebas que permitirá encontrar y detectar errores o defectos según sea el caso.

5.2.3.1. Definición de requerimientos

Una vez realizada la construcción del driver se definieron los siguientes requerimientos básicos:

- Abrir y establecer la comunicación del dispositivo (HMD).
- Terminar la comunicación, liberar el puerto y recursos comprometidos.
- Obtener información del estado actual de los sensores.
- Obtener el número de sensores del periférico.

Los anteriores son los requerimientos mínimos que debe soportar la librería y por lo tanto los drivers que se deseen implementar para otros dispositivos. Existen otros requerimientos funcionales que surgen a partir de la necesidad del usuario final (el programador) para configurar el dispositivo de acuerdo a condiciones especiales para una determinada aplicación de realidad virtual, estos se listan a continuación:

- Configurar la posición de referencia (posición cero) de cada uno de los sensores a utilizar en la aplicación.
- Configurar rangos máximos y mínimos de los sensores.

En la figura 28 se muestra el diagrama de casos de uso correspondiente a la definición de requerimientos listada anteriormente. La documentación de estos casos de uso se encuentra en el anexo D2.

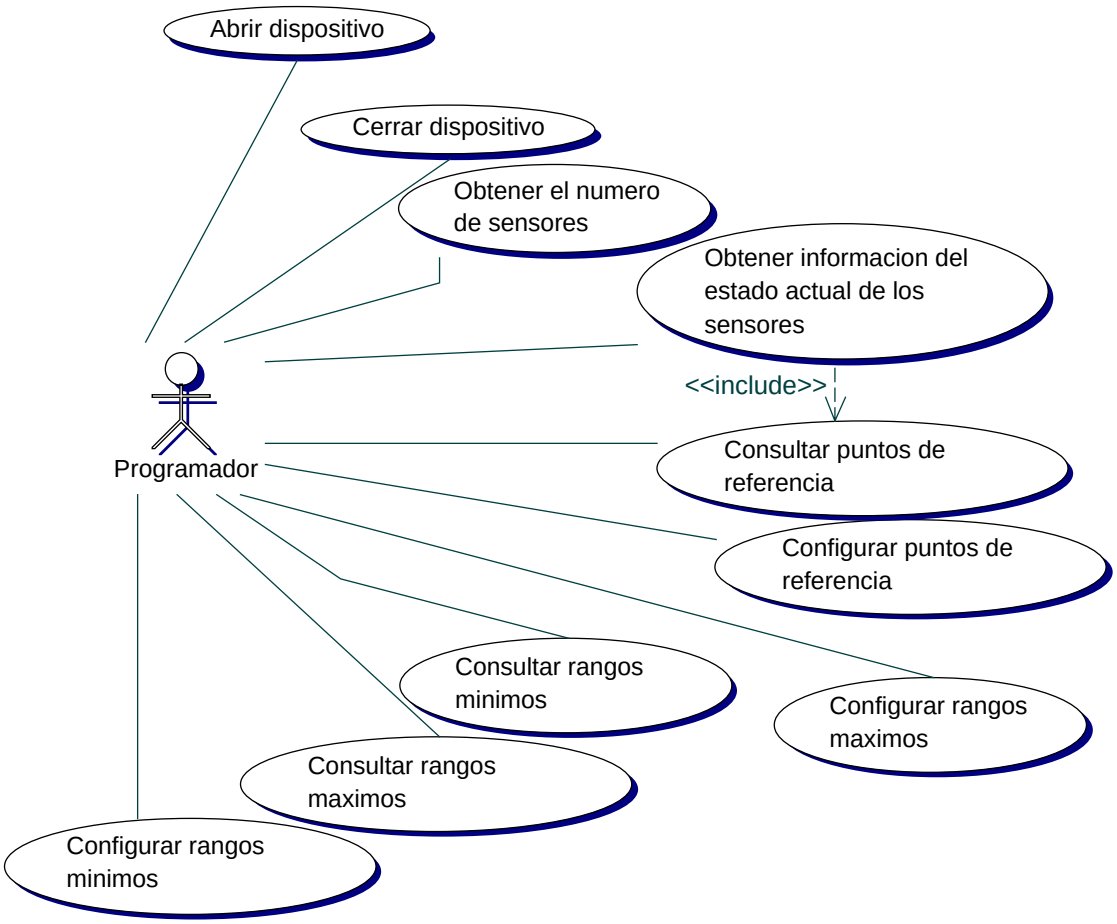


Figura 28 Casos de uso de la librería para cascos

5.2.3.2. Diseño

Una vez establecidos los requerimientos funcionales que debe suplir la librería para este dispositivo, se planteó el diagrama de clases que se muestra en la figura 29, en el cual se tiene una clase principal denominada “HMD” la cual utiliza una clase auxiliar “HMDConfiguration” para manejar la persistencia de los valores de configuración, además se cuenta con un patrón de diseño Factory Method para

administrar los drivers específicos de un determinado dispositivo.

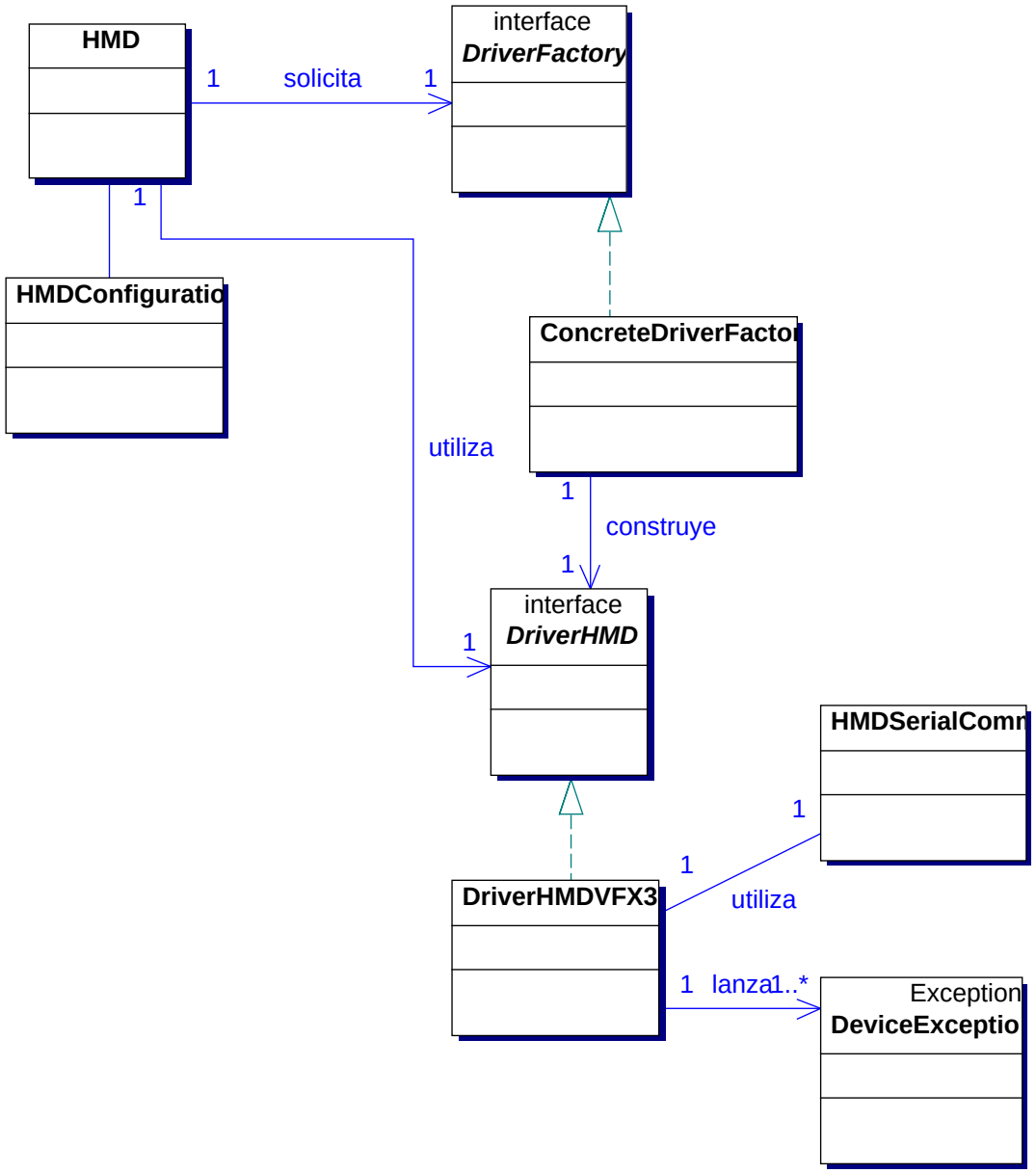


Figura 29 Diagrama de clases de la librería para cascos

A continuación se presentará una breve descripción de cada una de las clases que conforman la librería que administrará los HMD (Head Mounted Display), el diagrama de clases a nivel de implementación se muestra en el anexo B1.

- **HMD**: esta clase contiene la funcionalidad general que debe soportar la librería, es decir los métodos de abrir, cerrar y obtener la información actual del estado del dispositivo, además de los métodos de configuración de la

posición cero y de rangos máximos y mínimos, esta clase tiene asociaciones con **HMDConfiguration**, **DriverFactory** y **DriverHMD** las cuales se encargan de administrar la persistencia de la configuración, la administración de drivers y la comunicación con el dispositivo físico, respectivamente.

- **HMDConfiguration:** esta clase se encarga de administrar un archivo de propiedades en el cual se encuentra registrada de manera permanente la información concerniente a los puntos referencia (zero) y a los valores máximos y mínimos que soportarán los sensores en las aplicaciones a desarrollar.
- **DriverFactory:** esta interfaz define la estructura del patrón factory method que se utiliza para administrar los diferentes drivers que se pueden emplear en la librería, dichos drivers están definidos bajo la estructura de la interfaz **DriverHMD.**
- **DriverHMD:** esta interfaz define la estructura general a nivel de funcionalidad que deben soportar los drivers de acceso a los diferentes dispositivos físicos, esta funcionalidad esta regida por las características mencionadas en la sección **3.2.2.1.**
- **DriverHMDVFX3D:** esta clase constituye la esencia del driver construido para el casco VXF3D, el cual esta descrito en la sección **3.2.3,** esta clase tiene una asociación con HMDSerialComm que define los métodos nativos que utiliza el driver y que se mencionaron previamente.
- **DeviceException:** esta clase permite el manejo de excepciones dentro de la librería en general.

5.2.4. Proceso para la creación de la aplicación de configuración

Para la aplicación de configuración, se definieron los requerimientos y se diseño e implemento la aplicación, validando su correcto funcionamiento en la etapa de prueba, todos estos pasos se explican a continuación.

5.2.4.1. Definición de requerimientos

Para realizar la aplicación de configuración se tomó como base de ésta la aplicación que provee el fabricante para el mismo propósito, la cual permite básicamente lo siguiente:

- Configuración de la posición cero: establece como punto de referencia la posición actual de los sensores.
- Configuración de máximos y mínimos: permite que el usuario registre los rangos máximos y mínimos de alcance en cada uno de los ejes que maneja el dispositivo.

5.2.4.2. Diseño e implementación

Una vez definidos los requerimientos que la aplicación debe suplir, se tomó como base de diseño e implementación la interfaz gráfica que posee el programa que es suministrado por el fabricante y que se muestra en la figura 30.

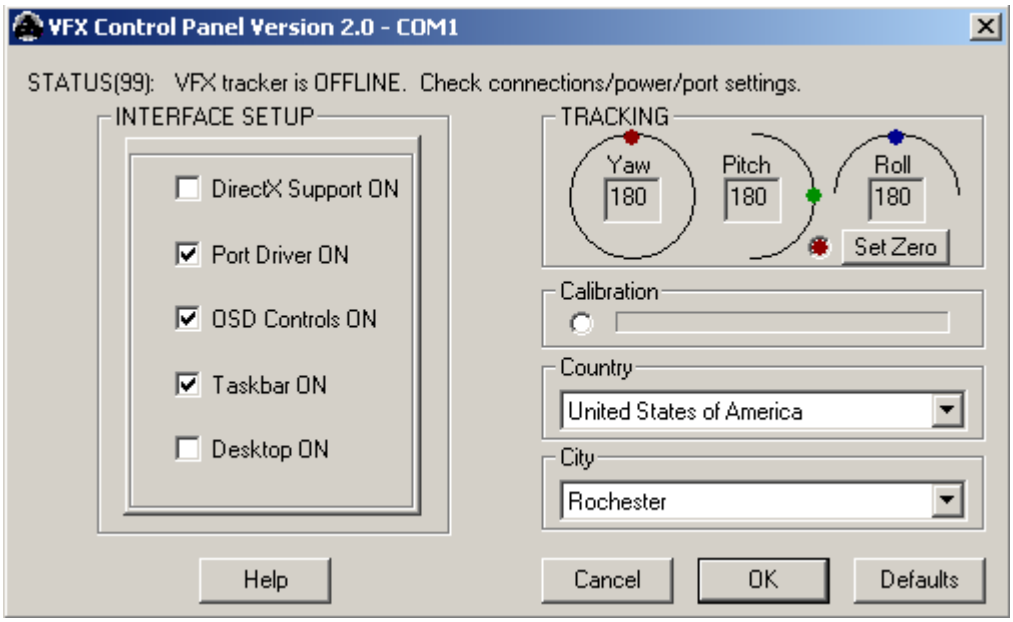


Figura 30 Aplicación de configuración del casco VFX3D suministrado por el fabricante

Posteriormente se construyó una aplicación Java, que cumple con los mismos objetivos, en la figura 31 se muestra la interfaz gráfica de dicha aplicación:

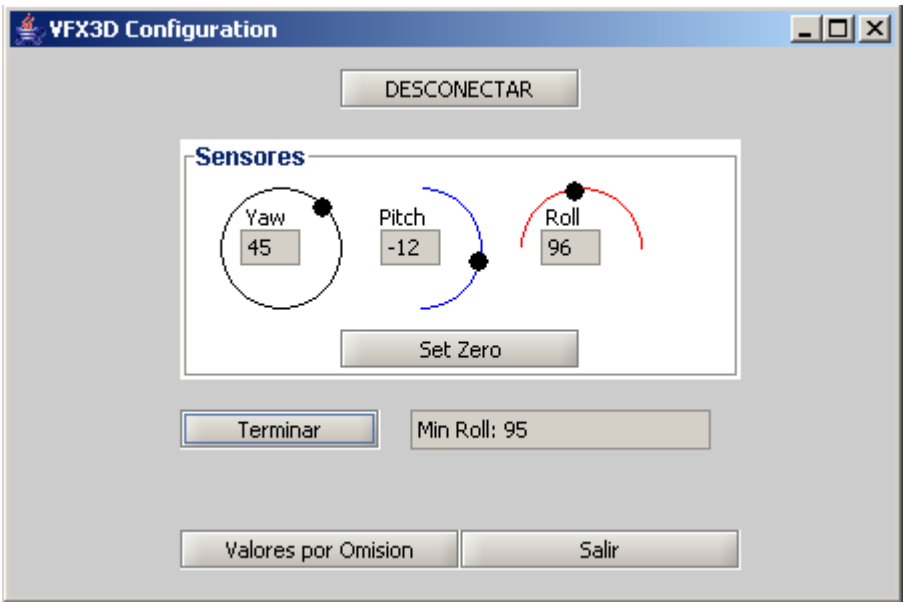


Figura 31 Aplicación Java de administración y configuración del casco VFX3D

5.2.4.3. Pruebas

Para realizar las pruebas generales de la librería se utilizó la aplicación de configuración mostrada en la sección anterior, los aspectos importantes a observar y analizar dentro de estas pruebas son los siguientes:

- Establecer y terminar la comunicación con el dispositivo: se analizó principalmente la funcionalidad que permite acceder al dispositivo, para verificar el proceso de comunicación. La aplicación respondió de manera adecuada al acceder al puerto serial y al liberar recursos comprometidos. Para iniciar la comunicación el botón en la figura 31 “DESCONECTAR” cumple la misma función que “CONECTAR”, dependiendo del estado de la comunicación este botón cambia de label.
- Información del estado actual de los sensores: uno de los aspectos más importantes es el cantidad de datos de los sensores que obtiene la aplicación desde el dispositivo, y la concordancia lógica entre el valor mostrado por la aplicación y la posición actual del dispositivo. Al realizar las primeras pruebas se observaron algunos problemas en cuanto a la concordancia de la información, esto se debía principalmente a la lógica de procesamiento en el driver, específicamente por que se estaba ignorando la relación entre los tres sensores que maneja el casco, es decir la información de cada uno de los sensores no debe verse de manera independiente, puesto que la posición actual de un sensor también varía cuando la posición de los otros sensores esta variando. Al solucionar este

error en el driver la aplicación funcionó correctamente y respondía naturalmente al movimiento del dispositivo.

- Tareas de configuración: en este aspecto se validó básicamente que la posición actual de los sensores se modificara al cambiar los puntos de referencia y los rangos máximos y mínimos, después de corregir algunos errores que surgían al determinar la posición correcta que debería devolver la librería, la funcionalidad de configuración respondió de manera adecuada. Para iniciar la configuración o guardado de los valores en los archivos de configuración el botón de “Terminar”, tiene la misma funcionalidad que “Iniciar” que cambia de label según el caso.

5.2.5. Conclusiones parciales

- El reconocimiento del protocolo de comunicación del casco, fue un proceso largo y demorado, donde se encontraron varias formas para su conocimiento, mediante la utilización de diferentes software, dando como resultado el descubrimiento del protocolo por parte de los investigadores.
- Los mecanismos de comunicación directa entre el dispositivo y el lenguaje java, no fueron posibles por la velocidad de transmisión a la cual envía el casco, esto trajo consigo la creación de un puente nativo para poder establecer dicha comunicación de manera exitosa.
- La librería presenta las características necesarias para poder expandirla, por la facilidad para agregar drivers de otros cascos, dentro del diseño de la librería, sin tener que cambiar el código en todas las clases involucradas en el proceso.
- La aplicación de configuración presentó una prueba importante sobre la lectura correcta de los datos, además de cubrir el requerimiento de guardar los datos máximos y mínimos necesarios y utilizados por el usuario de manera correcta.

5.3. Joysticks

A continuación se describirá el proceso de desarrollo de la librería que administra los joysticks.

5.3.1. Estudios Preliminares

Para estos estudios, se realizó la investigación de las especificaciones de joysticks y los mecanismos para establecer la comunicación con los mismos.

5.3.1.1. Estudio de especificaciones

Para poder definir los requerimientos se hizo necesario estudiar las especificaciones de diferentes tipos de joysticks, esto con la intención de construir esta librería de tal forma que abarcara el mayor número de dispositivos de este tipo, a continuación se describen los principales aspectos a tener en cuenta al analizar un joystick [39].

- Grados de libertad: representan básicamente el número de ejes en el que se puede mover el dispositivo, los posibles ejes son X, Y, Z, Yaw, Pitch y Roll, estos tres últimos son análogos a los tres grados de libertad que posee el casco VFX3D descrito en la sección anterior. Generalmente los joystick más sencillos poseen únicamente dos grados de libertad (X, Y) pero algunos más sofisticados poseen sensores que permiten abarcar otros grados de libertad.
- Tipos de conectores: representan el tipo de puerto por el que se conectan los joysticks. Existen varios tipos de conectores, pero los más comunes son el DB-15 (puerto de juegos)y el USB
- Número de botones: la cantidad de botones varía con cada tipo de joystick, sobre todo si es una palanca de juegos (por lo general 3 botones) o un gamepad (en general 6 o más botones).
- Force Feedback: algunos joystick poseen elementos que pueden crear vibraciones, resistencias de fuerza u otros elementos que proporcionan percepciones táctiles.

La Universidad Javeriana posee dos joysticks uno de ellos es un gamepad de dos grados de libertad y 10 botones, el otro es una palanca con tres grados de libertad (X, Y, Yaw) y dos botones, además maneja Force Feedback. Ambos dispositivos poseen conectores USB.

5.3.1.2. Definición del mecanismo de comunicación

Como se mencionó en el marco teórico, para el acceso a puertos USB desde una aplicación Java se estudiaron dos mecanismos JUSB y SDL para Java.

El primero de ellos se probó por medio de un ejemplos y su posterior modificación del código, con el objetivo de recibir o enviar algún dato, para establecer la comunicación con el dispositivo.

La consola que se muestra en la figura 32, contiene los datos de reconocimiento de los puertos USB, numerándolos y encontrando el nombre del dispositivo, su serial, la interface de USB que utiliza, además el tipo de Objeto al que pertenece.

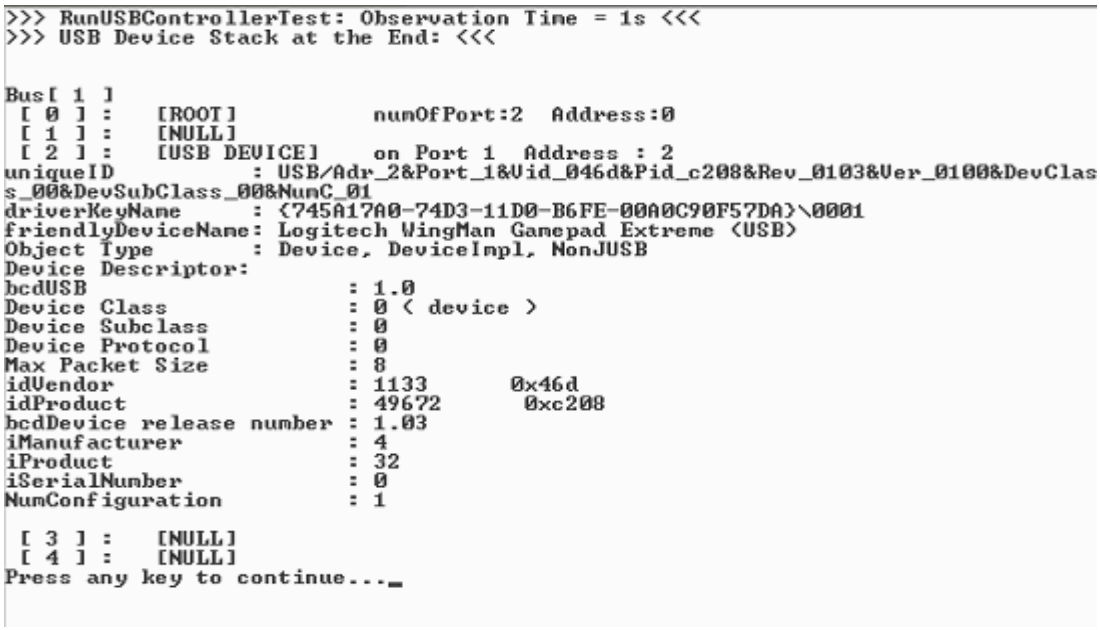


Figura 32 Prueba de JUSB

Al estudiar el API de la librería JUSB, se observaron algunas funcionalidades incompletas, de acuerdo al tipo de objeto en el que se calificaba al dispositivo, por esto no se pudo establecer la comunicación full dúplex.

A diferencia de JUSB, SDL para Java permite listar los joysticks conectados, listar el número de botones y de ejes y, lo más importante, obtener el estado actual de los mismos.

Como mecanismo de comunicación con joysticks USB se escogió a SDL para Java como la mejor alternativa, básicamente por el reconocimiento genérico de los dispositivos, lo cual ayuda a realizar la librería de manera idéntica para palancas y gamepads.

Debido a lo anterior no es necesario construir un driver específico para los joystick puesto que SDL se encarga de realizar las tareas básicas de acceso al periférico y la obtención de la información del estado actual de botones y ejes, por lo tanto el objetivo de la librería es encapsular la funcionalidad que proporciona SDL para darle un manejo de los periféricos mucho más sencillo y óptimo.

5.3.2. Proceso para la construcción de la librería

A continuación se muestran los pasos necesarios para la creación de la librería, como son la definición de los requerimientos, el diseño, la implementación y las pruebas para validarla.

5.3.2.1. Definición de requerimientos

Para determinar los requerimientos que debe suplir la librería se debe tener en cuenta que se trabajará sobre SDL, por lo tanto es importante mencionar la funcionalidad que este permite para delimitar los requerimientos. A continuación se mencionan los principales elementos que provee SDL:

- Enumerar los joystick conectados en el momento.
- Abrir un determinado joystick.
- Cerrar un determinado joystick.
- Obtener el número de ejes que posee el dispositivo.
- Obtener el número de botones que posee el dispositivo.
- Obtener el estado actual de un determinado botón.
- Obtener el estado actual de un determinado eje.
- Solicitar eventos de un determinado joystick.
- Obtener el número de Balls y Hats que posee el dispositivo, esto solo se aplica a cierto tipo de dispositivos denominados **trackballs,** pero esta funcionalidad no será tenida en cuenta para la definición de los requerimientos de la librería.

Una vez se determinaron las capacidades de SDL para el manejo de joystick se definió el diagrama de casos de uso que se encuentra en la figura 33.

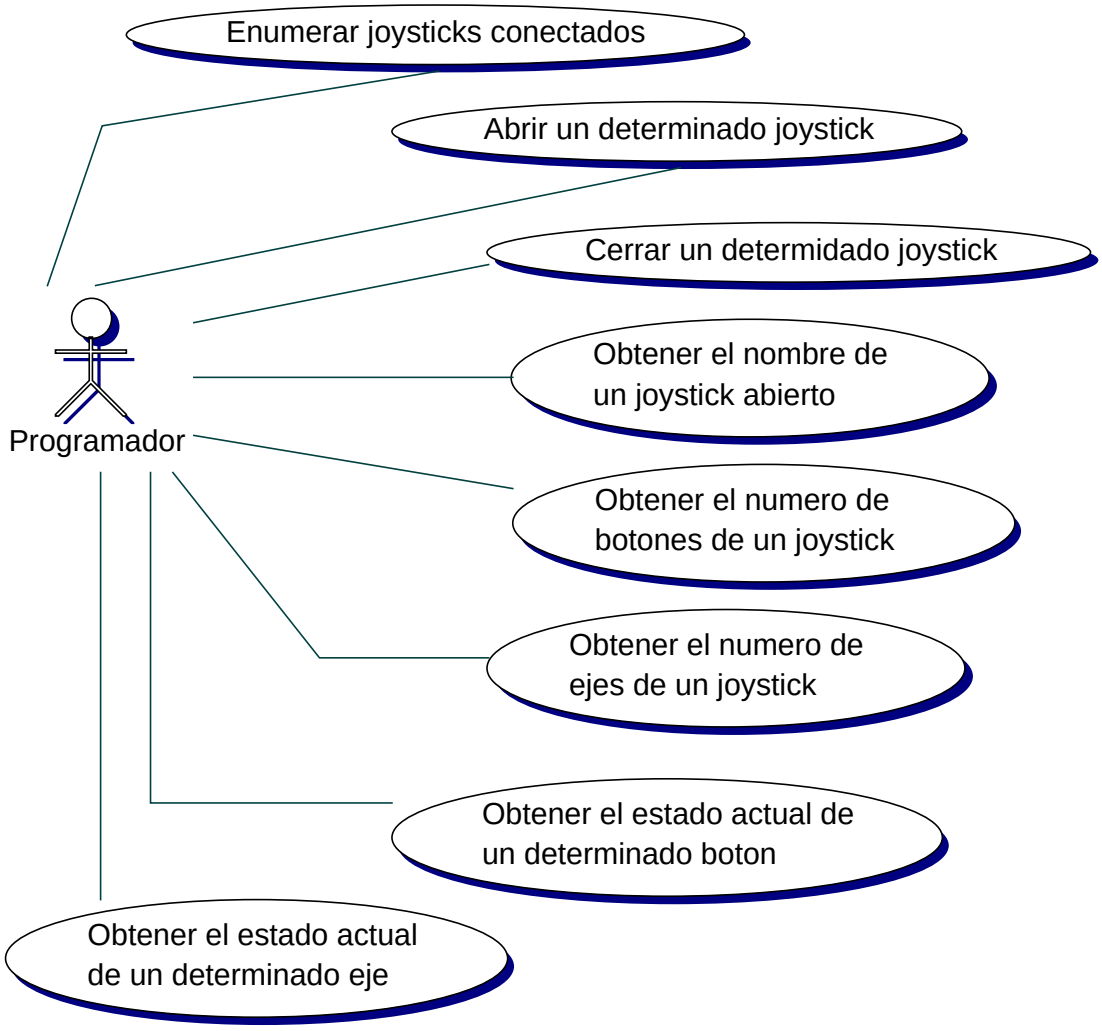


Figura 33 Diagrama de casos de uso de la librería del joystick.

5.3.2.2. Diseño de la librería

Con la definición de requerimientos de la sección previa se planteo el diseño de la librería para el joystick que se describe mediante el diagrama de clases de la figura 34.

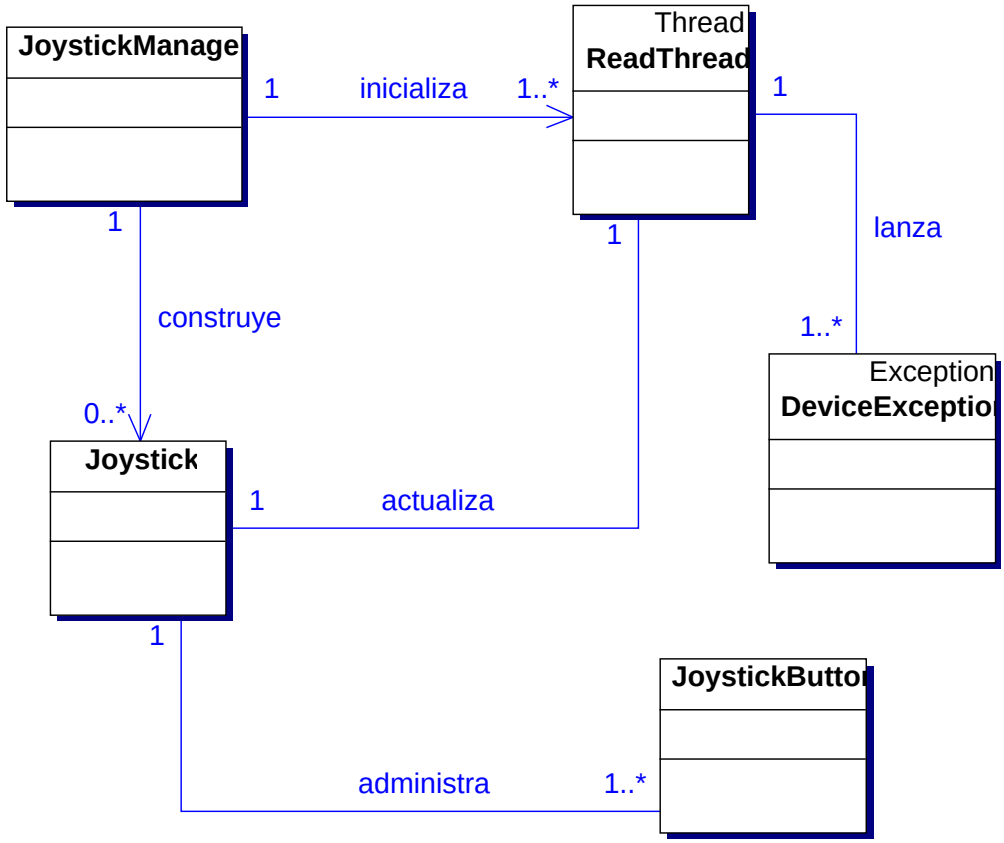


Figura 34 Diagrama de clases de la librería para joysticks

A continuación se presentará una breve descripción de cada una de las clases que intervienen en el manejo de los joysticks, el diagrama de clases a nivel de implementación se encuentra dentro del anexo B2.

- **JoystickManager:** Esta clase se encarga de la creación y administración de las instancias lógicas del joystick (clase Joystick), además inicializa un hilo (ReadThread) que esta actualizando el estado actual del joystick.
- **Joystick:** esta clase contiene la funcionalidad básica del dispositivo lógico, esto es específicamente la capacidad de listar el número de botones y ejes que posee el periférico físico y de obtener el estado actual de cada uno de los botones y ejes. Para el manejo de los botones se tiene un arreglo de instancias de la clase JoystickButton, el tamaño de este arreglo dependerá del número de botones que tenga el dispositivo físico.
- **ReadThread:** este hilo esta constantemente censando el periférico físico, para obtener el estado actual de cada uno de los ejes y botones, y posteriormente actualizar la instancia lógica (Joystick) del dispositivo.

- **JoystickButton:** esta clase permite administrar de manera sencilla el estado actual de cada botón.
- **DeviceException:** esta clase permite manejar de manera particular las excepciones que puedan ocurrir al administrar el dispositivo.

5.3.2.3. Pruebas

Para validar la coherencia del código y la funcionalidad en general de la librería se implementó una aplicación sencilla por consola, ver figura 35, en la que se enumeraban los joystick conectados y posteriormente establecía la comunicación con cada uno de ellos para mostrar los valores que correspondían al estado actual de ejes y botones, las pruebas fueron bastante buenas, los únicos parámetros que debieron ser ajustados fueron los tiempos de espera en el hilo que obtiene la información del estado actual, los cuales eran muy altos y se perdían algunos valores.

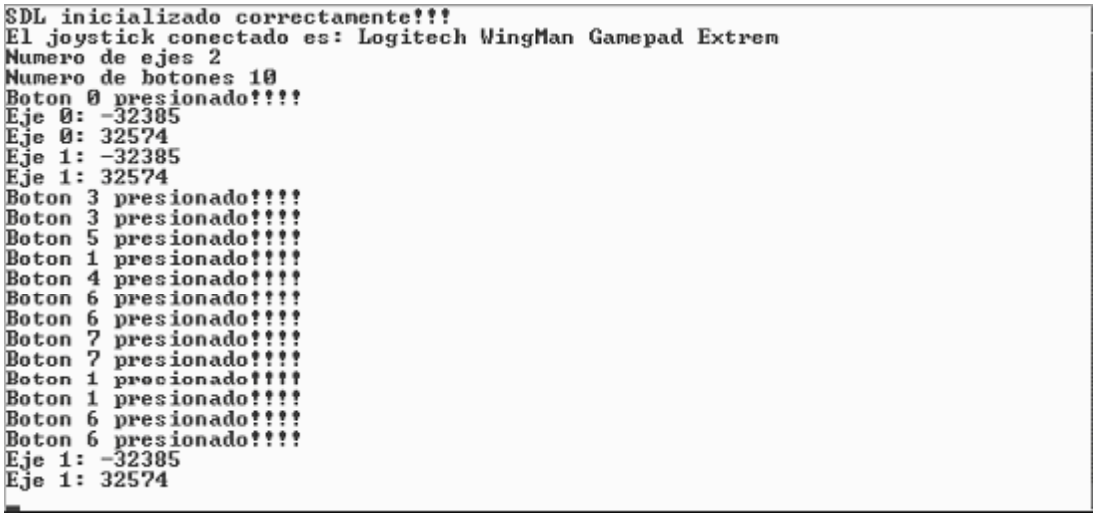


Figura 35 Prueba de implementación de la librería desarrollada para Joysticks

5.3.3. Conclusiones Parciales

- La librería SDL mejoró el proceso de construcción de la librería, debido a la facilidad de manejo y reconocimiento de los dispositivos conectados al puerto USB, sin restricciones en el tipo de dispositivo conectado, gamepad o joystick.

5.4. Diseño general de la librería

Una vez se construyeron las librerías descritas anteriormente, se realizaron actividades de evaluación, para determinar la posibilidad de unificar todas las

implementaciones y darle un manejo más sencillo, esto para facilitar la tarea del usuario final, o sea el programador.

5.4.1. Estudio y definición de patrones de diseño

Para poder unificar la librería una de las tareas primordiales era el estudio de patrones de diseño que facilitaran dicho trabajo, pero para realizar esto era necesario evaluar en primera instancia las características propias de los dispositivos físicos y posteriormente las características de las implementaciones particulares de cada uno de ellos.

Estudiando las especificaciones de los dispositivos y las características de las librerías desarrolladas hasta el momento, se determinó que la administración de los joysticks se realizaría de manera independiente al resto de la librería. Lo anterior se decidió, con base en la estructura propia de este tipo de dispositivos (ejes y botones), el cual dista mucho de la estructura física de los cascos y guantes de datos (basada en sensores de movimiento), además las características de la librería de joysticks no se acoplan lógicamente a los otros dos tipos de periféricos.

Debido a lo anterior se estudiaron algunos patrones de diseño para acoplar la librería de cascos y guantes de datos, básicamente se estudiaron los siguientes patrones:

- Proxy
- Abstract Factory
- Factory Method

Posteriormente se evaluaron las necesidades que debían suplir dichos patrones, las cuales se listan a continuación:

- Manejo de drivers: esto para poder emplear diferentes drivers para dispositivos específicos y no atar la librería solamente a los dispositivos que posee la Universidad Javeriana.
- Construcción de instancias especificas: el manejo de instancias de dispositivos lógicos debe manejarse de manera transparente y unificada.

Para suplir estas necesidades se decidió emplear el patrón Factory Method para poder construir y manejar los drivers, en este caso los drivers **DriverGlove** y **DriverHMD.** Para la construcción de instancias de los dispositivos lógicos se seleccionó el patrón Abstract Factory como se puede apreciar en la figura 36.

Una vez definidos los patrones se aplicaron dentro del diseño general de la librería, como se puede apreciar en el diagrama de clases de la figura 36, se

pueden comparar estos patrones de acuerdo al anexo E, que permite ver la estructura principal de dichos patrones.

Se creó una nueva clase denominada **Device** la cual define el comportamiento general de los dos tipos de dispositivos (cascos y guantes de datos), de manera tal que el programador siempre va a utilizar esta clase y no **DataGlove** o **HMD,** también es importante observar que el manejo de los drivers se unifica al utilizar el factory method.

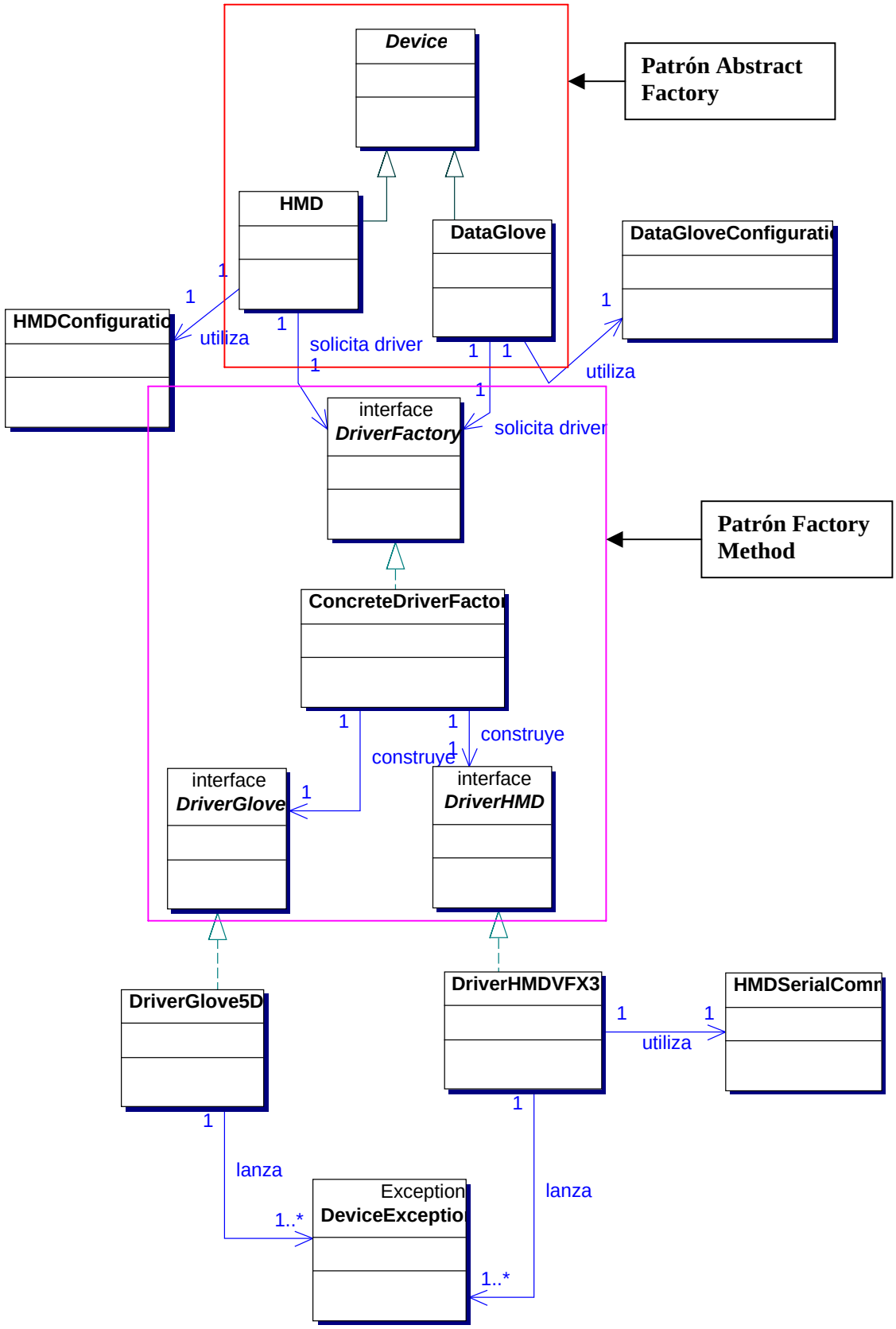

Device
Patrón Abstract Factory
HMD
DataGlove
DataGloveConfiguratio
HMDConfiguratio
1
utiliza
solicita driver
interface
DriverFactory
Patrón Factory Method
ConcreteDriverFactor
construye
interface
DriverGlove
interface
DriverHMD
DriverGlove5D
DriverHMDVFX3
HMDSerialComm
lanza
Exception
DeviceExceptio
1..*

Figura 36 Diagrama de clases luego de haber aplicado patrones de diseño

En la figura 36 se observan dos cuadros, que enmarcan los patrones de diseño utilizados, el de color rojo es el Abstract Factory y el morado es Factory Method.

5.4.2. Estudio y definición de mecanismos de comunicación con dispositivos de entrada lógicos

Una de los aspectos fundamentales a tener en cuenta en el desarrollo de la librería es el mecanismo mediante el cual el programador obtiene la información desde el dispositivo lógico, puesto que las aplicaciones de realidad virtual son complejas y pesadas consumen bastantes recursos del computador para realizar tareas esenciales como lo es el renderizado, por lo tanto si la librería llegase a ser una carga pesada para la máquina, las aplicaciones pueden verse afectadas seriamente.

En un principio la librería estaba diseñada para que el usuario (programador) solicitara la información del estado actual del dispositivo, lo cual hace que tenga que manejar la sincronización de los datos y el manejo de hilos de procesamiento, además el usuario puede llegar a caer en el error de realizar el renderizado cada vez que se lee la información del dispositivo, pero esto hace que un proceso tan pesado como el de pintado se realice constantemente independiente de si se registró cambio en el estado del dispositivo físico.

Para dar una mayor libertad al programador se estudiaron otros mecanismos de comunicación, específicamente en el modelo de VR-Juggler y en el de GKS (ver marco teórico) siendo este último, el modelo elegido para la librería, esta decisión se tomó básicamente por las facilidades en cuanto a diseño e implementación, además el Kernel de VR-Juggler es demasiado complejo de entender y por lo tanto costoso al momento de implementar.

Como se describió en el marco teórico, en la especificación de GKS existen tres modos diferentes para establecer la comunicación entre dispositivos lógicos, Request, Sample y Event, se determinó que se implementaría el modo Event, debido a la existencia dentro de la librería del modo Request.

Una vez implementado el modo de comunicación por eventos, se realizaron las pruebas del caso acorde al nuevo ciclo de implementación desarrollado, los resultados fueron bastante satisfactorios, posteriormente se evaluaron los pros y los contras de este nuevo modelo, y se decidió modificar la forma en que los eventos llegan hasta la aplicación.

La modificación en la forma de los eventos, se basa en que los eventos son **solicitados,** lo cual hace que el programador se adapte a un modelo bastante extraño en el lenguaje de programación Java, por lo tanto se implantó el modelo que se aplica a los componentes gráficos de Swing el cual es por **notificación,** es decir que al generarse un nuevo evento este se notifica directamente a un objeto previamente registrado por el programador.

La aplicación de este nuevo modelo definió el diseño final de la librería el cual se muestra en la figura 37.

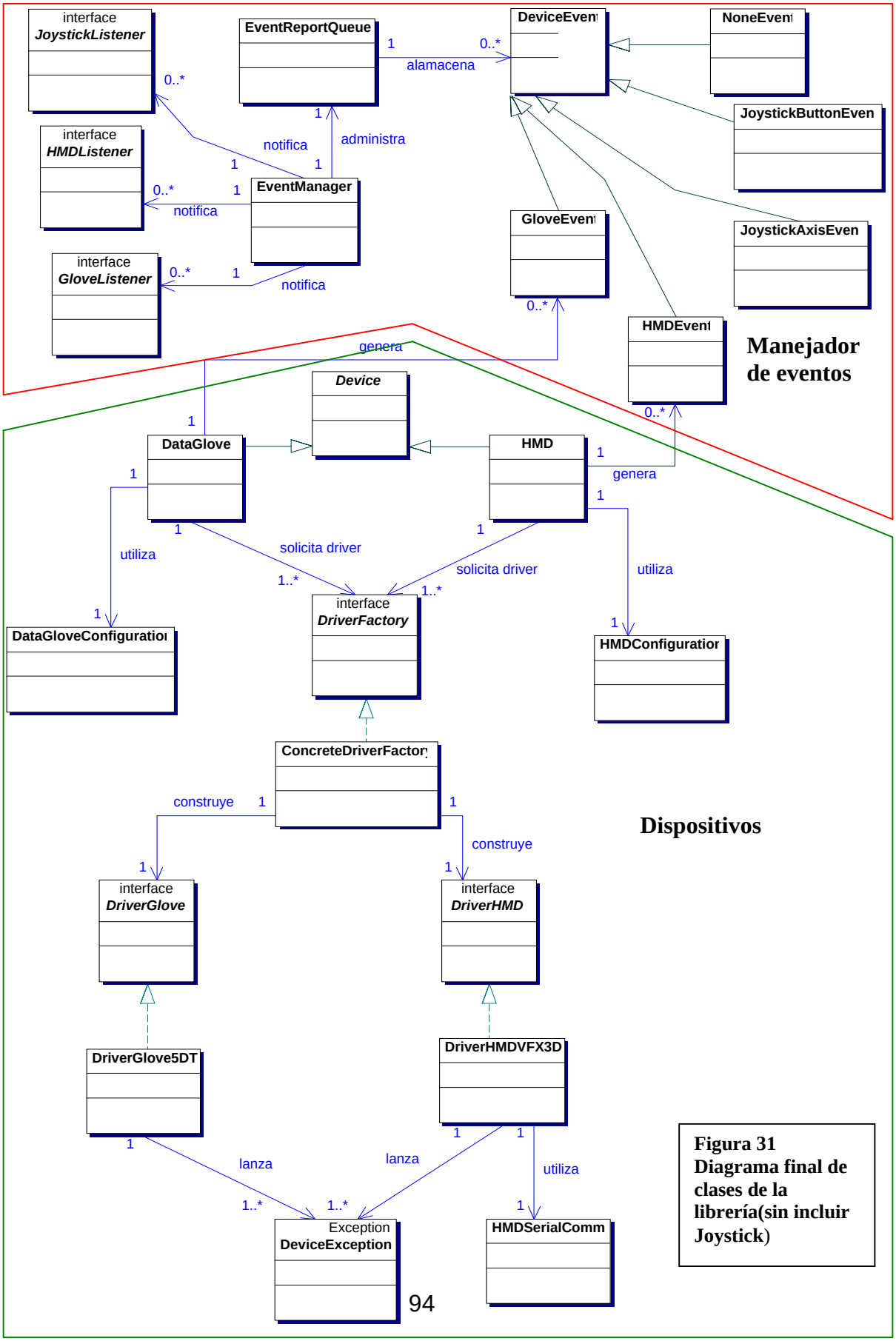


Figura 31
Diagrama final de clases de la librería(sin incluir Joystick)

Figura 37 Diagrama final de clases de la librería (sin incluir joystick)

El manejo de joysticks también se vio modificado la figura 38 muestra el diagrama de clases correspondiente.

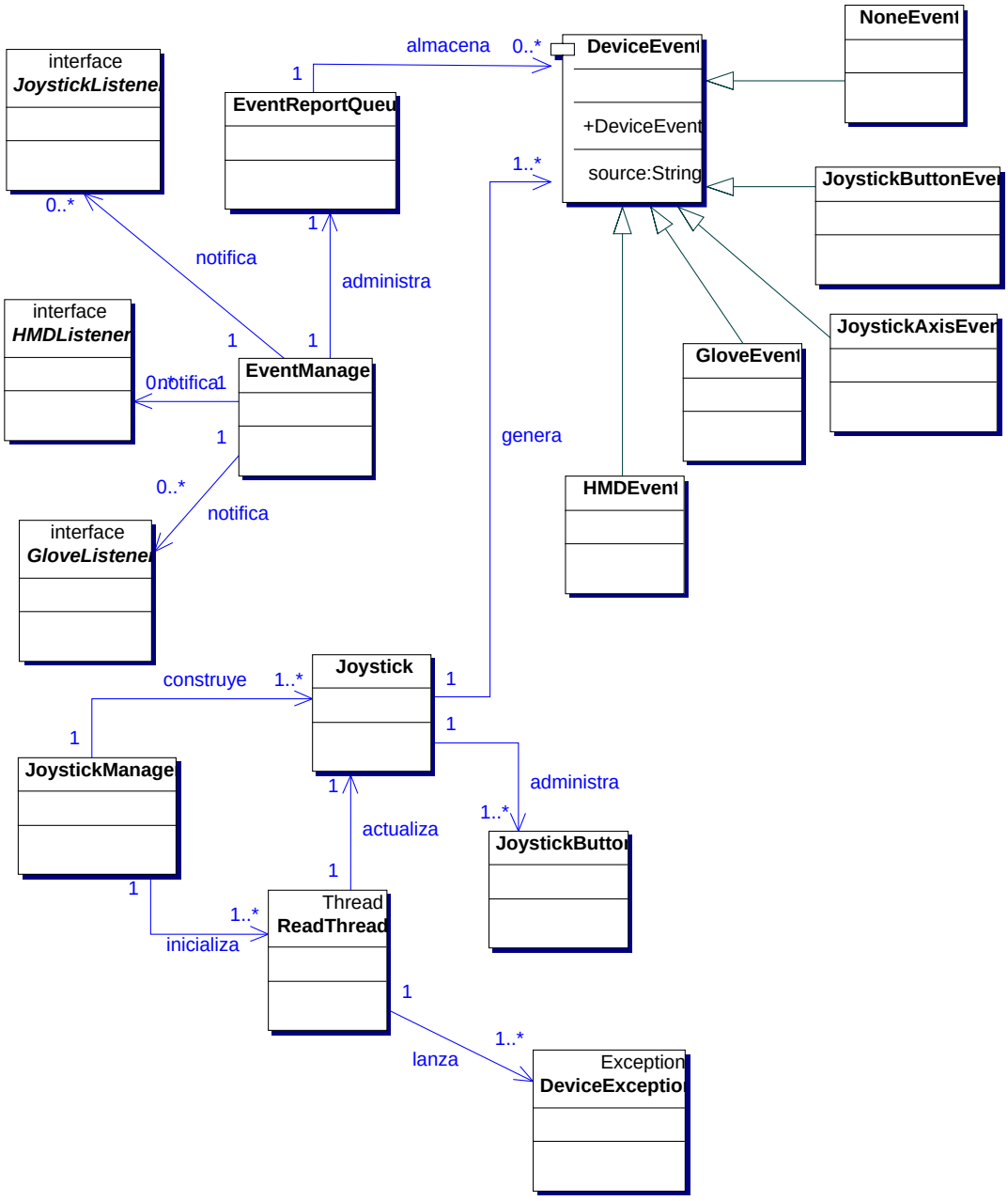


Figura 38 Diagrama de clases para la librería de administración de joysticks

6. PRUEBAS Y RESULTADOS

En esta sección se describirá el plan de pruebas, su ejecución y el análisis de resultados obtenidos.

6.1. Plan de pruebas

El principal objetivo del plan de pruebas es determinar el tipo de prueba adecuado, para validar el funcionamiento de la librería y determinar si se encuentra entre los mínimos requeridos para un entorno gráfico de realidad virtual. Para realizar lo anterior se mencionarán las variables a evaluar dentro de las pruebas, los riesgos asociados con la ejecución de las pruebas y la descripción de requerimientos mínimos a nivel de hardware y software.

6.1.1. Justificación y misión de la prueba

Las aplicaciones gráficas de realidad virtual se caracterizan principalmente por la exigencia de un hardware robusto que permita un desempeño adecuado de todas las tareas involucradas en la ejecución de las mismas, por lo tanto se hace necesario observar el comportamiento de la librería desarrollada en un ambiente de realidad virtual y analizar el acoplamiento entre los dispositivos de entrada utilizados y el entorno Java.

El principal objetivo de la prueba es determinar si las aplicaciones gráficas desarrolladas en Java, pueden ser manipuladas desde dispositivos de entrada físicos y analizar el comportamiento de dichas aplicaciones, al ser manipuladas y transformadas por dichos periféricos.

6.1.2. Variables a ser evaluadas

Desde el inicio de la investigación, se planteó un serio interrogante acerca de la posibilidad de desarrollar aplicaciones Java, que pudieran trabajar con diferentes dispositivos de realidad virtual de manera conjunta y acoplada sobre un mismo entorno gráfico, por lo tanto la principal variable que se piensa medir y evaluar es el funcionamiento de la librería al trabajar con más de dos dispositivos a la vez dentro de aplicaciones gráficas Java. Este funcionamiento se puede validar mediante la concordancia entre los valores obtenidos por la lectura de los dispositivos y la respuesta del mundo virtual a dichos valores, también se evaluarán los tiempos de administración de la información que la librería realiza para determinar si el desempeño de la misma depende de la complejidad de las aplicaciones gráficas que la empleen.

Un punto importante dentro de la librería es la portabilidad a otros sistemas operativos por lo tanto se realizarán pruebas que permitan determinar a que sistemas operativos es extensible la librería desarrollada.

6.1.3. Requerimientos de la aplicación de prueba

Para evaluar el correcto funcionamiento de la prueba se determinaron los siguientes requerimientos que la aplicación debe suplir:

- Movimientos de la cámara: La aplicación deberá soportar movimientos de cámara en primera persona, es decir, la cámara será una abstracción del punto de vista de una persona inmersa en el mundo, por lo tanto la cámara debe poder proporcionar los siguientes movimientos:
    - Rotación sobre su propio eje, es decir este movimiento debe ser análogo al de mover la cabeza de izquierda a derecha y viceversa.
    - Arriba y abajo, es decir la cámara debe permitir observar objetos que estén más arriba o más abajo del campo de visión inicial.
    - Desplazamiento horizontal, es decir la cámara deberá simular movimiento lateral y frontal dentro del mundo.
- Interacciones: La aplicación deberá permitir que el usuario pueda realizar algún tipo de modificación en el mundo virtual, estas modificaciones pueden ser cambiar los colores de un objeto, rotarlo o desplazarlo a voluntad dentro del ambiente virtual.

6.1.4. Riesgos

Dentro de las pruebas preliminares de la librería se había probado el comportamiento independiente de cada unos los dispositivos y el resultado fue óptimo, pero nunca se había empleado la librería con más de un dispositivo a la vez por lo tanto el principal riesgo durante la ejecución del plan de pruebas radica en el acoplamiento de los periféricos en una aplicación Java, pues podrían surgir inconsistencias en el momento en que todos los dispositivos intenten modificar el mundo virtual, afectando directamente la aplicación.

6.1.5. Características de la pruebas

Básicamente la prueba general consiste en la realización de un ambiente de realidad virtual que permita la interacción del usuario dentro del mismo, dicho ambiente puede tener la capacidad de administrar dispositivos de entrada, el renderizado del mundo y los eventos que sean generados por el usuario; a

continuación se describirán los equipos empleados en las pruebas, la aplicación virtual y posteriormente la descripción de la prueba general.

6.1.5.1. Descripción de los equipos

Para la realización de las pruebas se empleó un computador con las siguientes características:

Sistema operativo	MS Windows 2000, Linux Fedora
Procesador	Intel ® Pentium IV CPU 1,70 GHz
Modelo del sistema	Compaq EVO D300
Memoria RAM	256 MB
Puertos USB	2
Puertos Seriales	2
Tarjeta de video	Nvidia Vanta 16MB

También se realizaron pruebas sobre equipos de la empresa SUN Microsystems con sistema operativo Solaris 9.0 y procesador SPARC.

6.1.5.2. Descripción de la construcción de la aplicación virtual

La aplicación virtual, se creó en **OpenGL** para java, esta librería se instaló de acuerdo a las indicaciones encontradas en la pagina web de ***jausoft***[25], después de tener la instalación completa, se prosiguió a la creación del mundo en un editor gráfico de 3D, como es el **3D studio max**, en el cual se generó un archivo VRML que después se exportó al formato OBJ para finalmente crear un archivo de texto, cambiando unos caracteres (/ por un espacio) para su correcta lectura.

Después de la creación del archivo, se creó una aplicación, con una clase ***Canvas*** privada, que permite realizar el pintado, creación de la ventana y los correspondientes manejadores de eventos, como el teclado, mouse, casco, guante y joystick.

El mundo virtual, tiene 2 componentes gráficas principales, la creación del fondo, que es un cubo, con la textura de un paisaje, y la aldea en el medio que se carga del archivo de texto; para las texturas se utilizo una función llamada LoadTextures, donde se envía el nombre del archivo, donde solo se cargan archivos .png (ver figura 39).

El mundo permite la navegación en todos los sentidos, de rotación, de traslación, de esta forma se puede probar los datos y su interpretación por parte del programador, de acuerdo al movimiento precisado por el dispositivo.

Figura 39 Vista inicial de la aplicación virtual

6.1.5.3. Descripción de la prueba general

La prueba tiene dos partes, en la primera de ellas se conecta cada dispositivo de manera independiente para poder interactuar con el mundo, en la aplicación se configuró el casco para el manejo de la rotación de la cámara, con el guante y el joystick se realizan los desplazamientos y rotaciones de la cámara de manera análoga al casco.

Con el guante de la mano derecha, se implementó los movimientos de la siguiente manera:

- Con el dedo pulgar se traslada la cámara hacia adelante.
- Con el índice se traslada cámara hacia atrás.
- Con el anular se rota la cámara hacia la izquierda.
- Con el meñique se rota la cámara hacia la derecha.

Si se utiliza el guante de la mano izquierda se intercambian las funciones de esta forma:

- Con el dedo meñique se traslada la cámara hacia adelante.
- Con el dedo anular se traslada la cámara hacia atrás.
- Con el dedo índice se rota la cámara hacia la izquierda.
- Con el dedo pulgar se rota la cámara hacia la derecha.

Para el joystick, los movimientos fueron implementados con el movimiento de los ejes, de la siguiente manera.

- El eje y hacia arriba se traslada la cámara hacia adelante.
- El eje y hacia abajo, se traslada la cámara hacia atrás.
- El eje x, permite rotar la cámara, hacia a la izquierda y hacia la derecha respectivamente.

La segunda parte consiste en conectar los tres dispositivos diferentes que sean administrados por la librería implementada, dentro de una aplicación de realidad virtual desarrollada en OpenGL para Java, tales dispositivos son específicamente el casco virtual, el guante de datos y un gamepad, la aplicación java creada presenta la interacción con el teclado, de acuerdo a los Listeners que facilita el manejador de eventos de Java. Cada uno de los anteriores dispositivos cumple una función diferente dentro del mundo virtual a continuación se describen dichas funciones:

- Casco de realidad virtual: Se encargará de realizar los movimientos rotacionales propios de la cámara, es decir los movimientos de la vista del usuario dentro del mundo.
- Gamepad: Se encargará de realizar los desplazamientos dentro del mundo, de acuerdo a la vista en la que se encuentre, hacia delante o hacia atrás.
- Guante de datos: con este dispositivo también se realizarán desplazamientos dentro del mundo, hacia delante o atrás.

De acuerdo a estos movimientos, se planea dar la vuelta a la casa, utilizando los dispositivos, observando las variaciones y tomando los tiempos de administración de eventos, es decir desde que se genera el evento hasta que este llega a la aplicación justo antes de ser procesado. Para poder establecer un punto de comparación en cuanto a los tiempos de administración de eventos se realizaran estas mediciones no solo con un mundo virtual sino con diferentes aplicaciones gráficas que presenten diversas complejidades a nivel gráfico especialmente en el renderizado.

Después de estas pruebas se prosiguió a realizar la verificación del funcionamiento de la librería para el casco y guante en diferentes sistemas operativos, para el guante se realizo en Solaris y para el gamepad en Linux.

Para la ejecución del plan de pruebas se especificaron las siguientes actividades:

- Instalación de software y librerías necesarias
- Instalación de la librería de administración de dispositivos de entrada de realidad virtual

- Instalación de los drivers necesarios para los periféricos empleados
- Instalación y ejecución de las aplicaciones de configuración
- Actividades de programación y procesamiento de eventos de los dispositivos.
- Ejecución de la aplicación.
- Medición de tiempos de administración de eventos.
- Pruebas de la librería en sistemas operativos Linux y Solaris.

6.2. Ejecución del plan de pruebas

Una vez implementada la aplicación de prueba que satisface los requerimientos descritos, se prosiguió a la ejecución de la prueba, mostrando los siguientes resultados.

- En la sección 6 se describirá el proceso de instalación de las librerías, necesarias para el correcto funcionamiento.
- Instalación de la librería de administración de dispositivos de realidad virtual: en esta fase se instalaron los archivos correspondientes a la librería implementada y los archivos de texto que almacenan los datos de configuración.
- Instalación de los drivers necesarios para los periféricos empleados: en esta etapa se instalaron los driver del guante de datos y del casco virtual, para que las aplicaciones java puedan acceder a los dispositivos físicos. En esta fase no se presentaron inconvenientes mayores.
- Instalación y ejecución de las aplicaciones de configuración: en esta etapa se instalaron las aplicaciones de configuración para el guante de datos y el casco virtual. Posteriormente se ejecutaron las aplicaciones pero se presentaron problemas al acceder a los archivos de configuración, ya que en la pruebas preliminares de dichas aplicaciones los archivos eran accedidos de manera local, es decir dichos archivos estaban en el mismo lugar que los archivos compilados, por lo tanto al empaquetar la librería y dejar los archivos de configuración fuera del archivo JAR que la contiene, las aplicaciones no podían encontrar tales archivos. Una vez corregido este defecto de la librería las aplicaciones funcionaron correctamente.
- Actividades de programación y procesamiento de eventos de los dispositivos: en esta etapa se realizaron los ajustes de código dentro de la aplicación de prueba para que esta accediera y administrara los dispositivos de entrada, dichos ajustes son básicamente la implementación de una serie de interfaces denominadas *Listeners* para que la aplicación pueda ser notificada al generarse un evento en cualquiera de los dispositivos conectados, en la sección 6 se describirá en detalle la utilización de la librería.

6.2.1. Primera prueba

La primera prueba tiene por objeto, reconocer los movimientos dentro del mundo de cada uno de los dispositivos de manera independiente, para reconocer el correcto funcionamiento dentro de la librería de cada uno de los drivers.

Para esta prueba se decidió ir capturando cada uno de los frames generados al realizar un cambio en el entorno (por cada variación en el dispositivo analizado), el orden de estas pruebas fue, casco, guante y joystick.

- La figura 40 muestra el inicio de la aplicación, donde sale de frente la casa del mundo virtual.

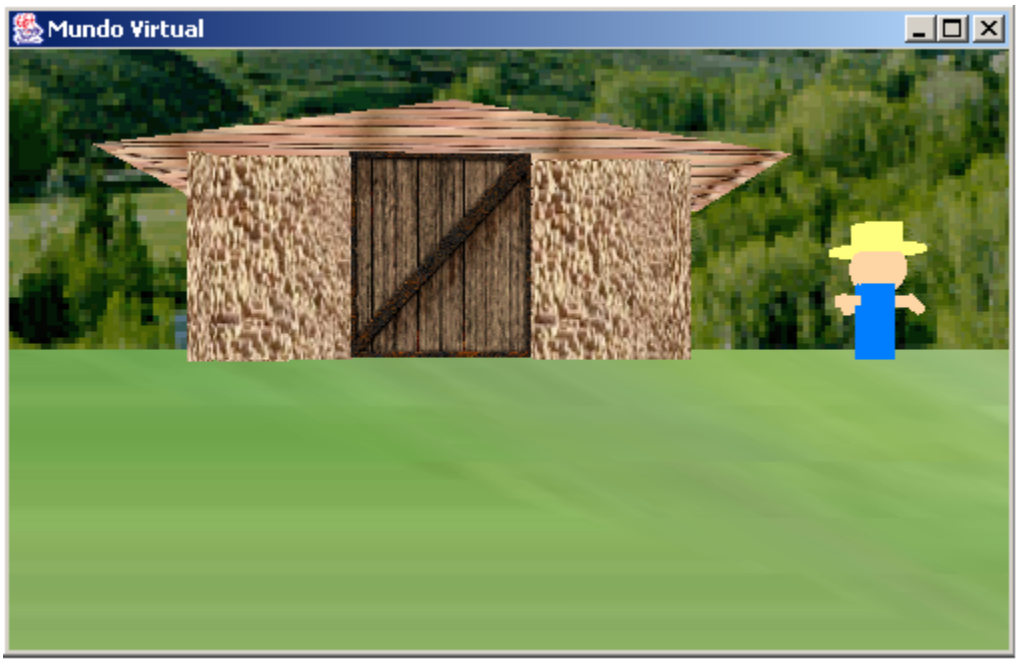


Figura 40. Posición inicial en la prueba del casco

- Después en la figura 41, se observa el cambio en la vista del mundo, debido al movimiento del casco hacia la izquierda.

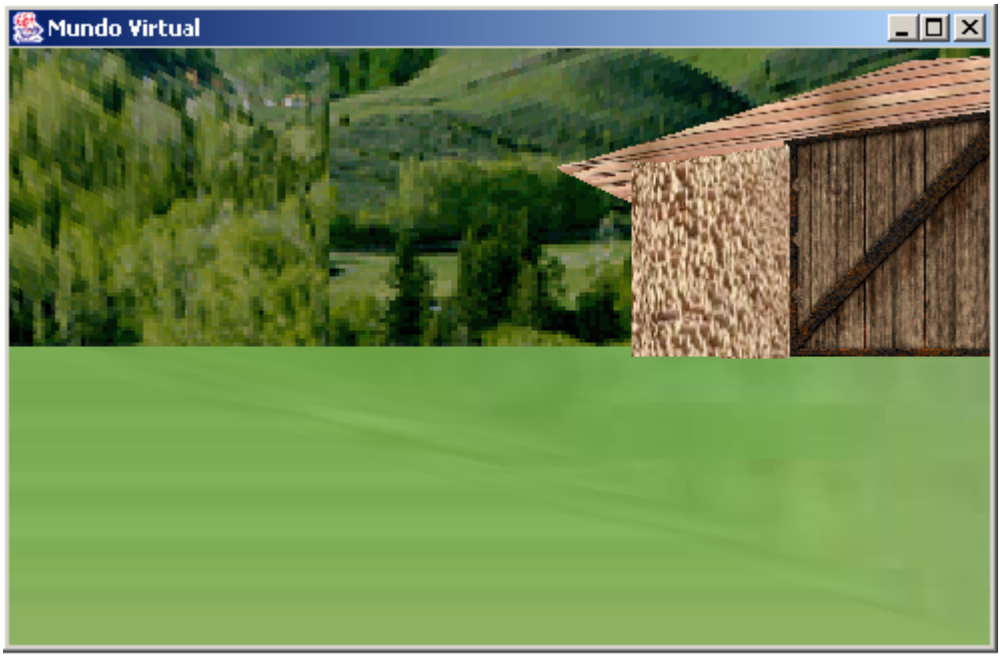


Figura 41 Vista del mundo después del primer movimiento del casco

- En la figura 42 se muestra el cambio de la posición con el movimiento del casco hacia la derecha, dependiendo del cero configurado por el usuario.

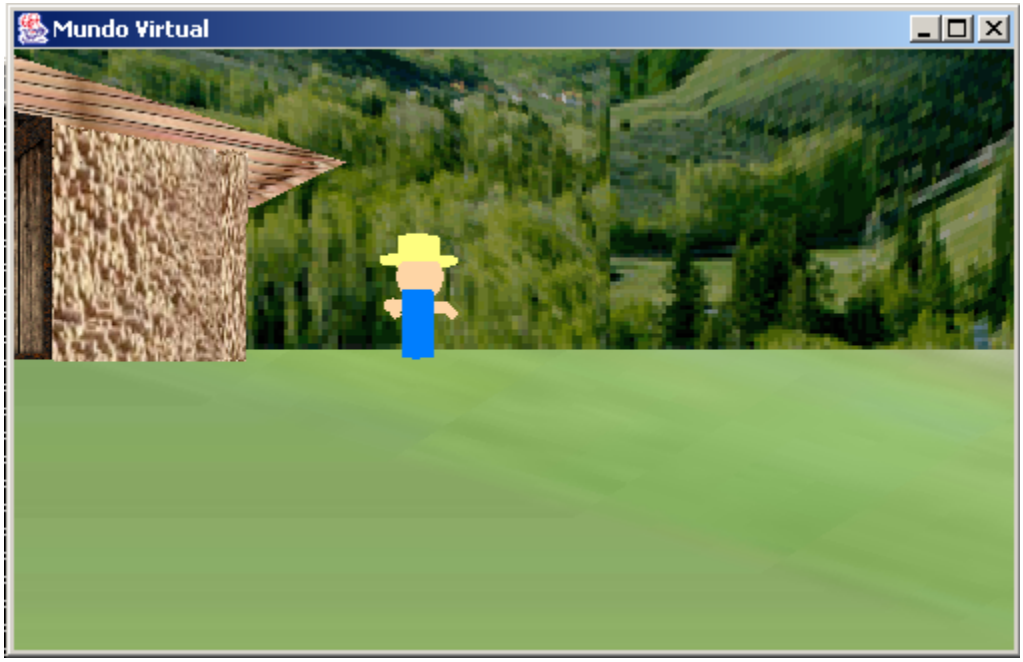


Figura 42 Vista del mundo después del segundo movimiento del casco

- En la figura 43, se ve la posición inicial del mundo cuando se probo el guante de datos.

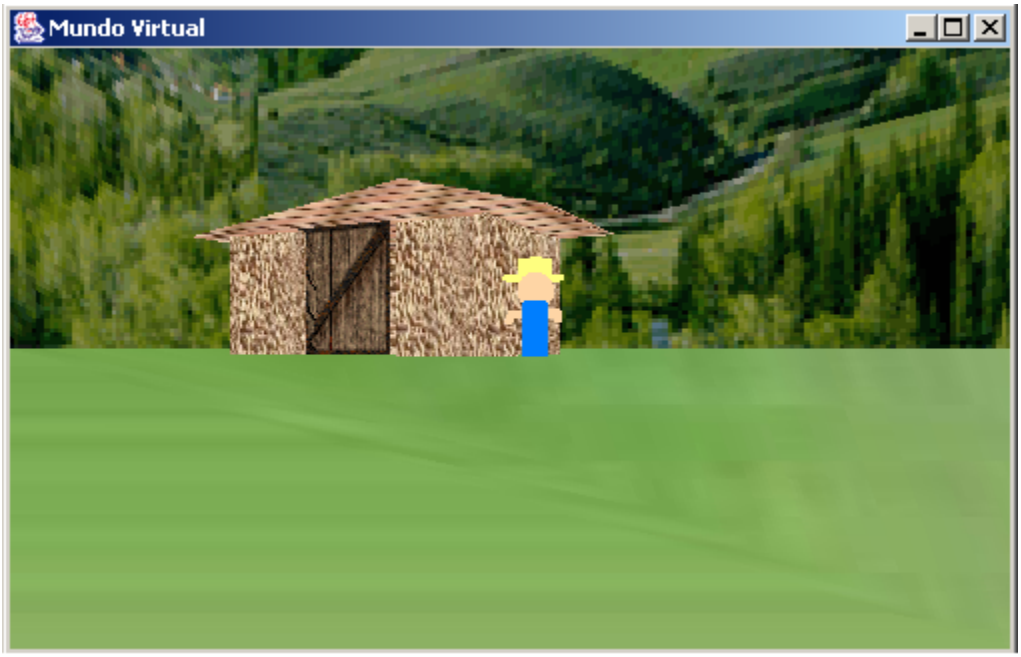


Figura 43 Posición inicial para la prueba del guante

- En la figura 44, se observa cuando se aleja la cámara, al mover el dedo respectivo (índice, mano derecha) para esta tarea.

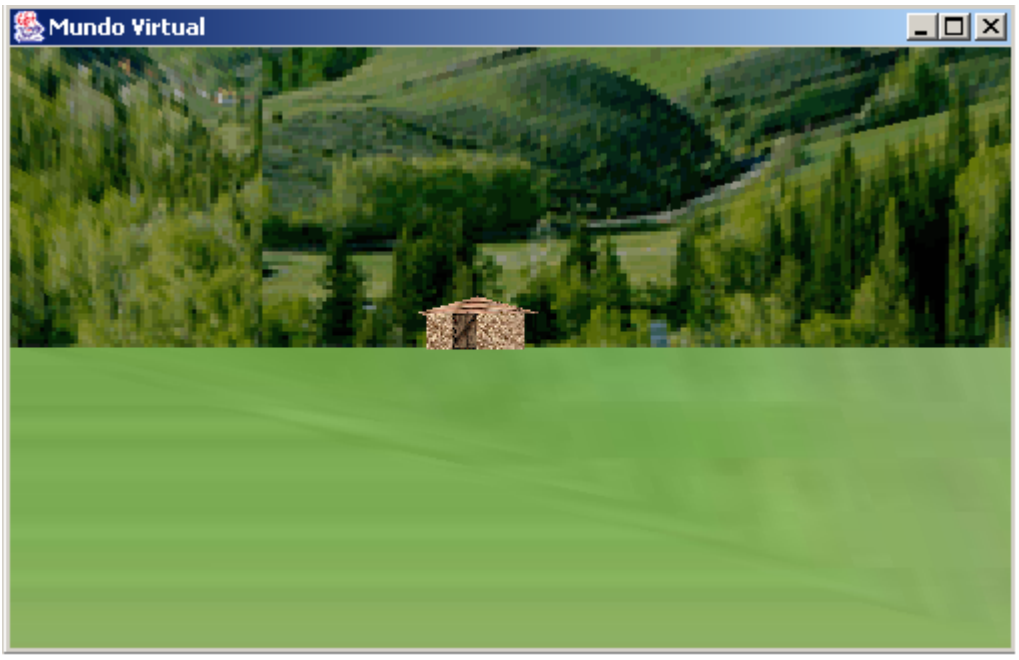


Figura 44. Posición después alejarse la cámara de acuerdo a la posición anterior

- En la figura 45, se observa cuando la cámara se acerca de acuerdo a la posición anterior, al mover el dedo indicado (pulgar derecho).

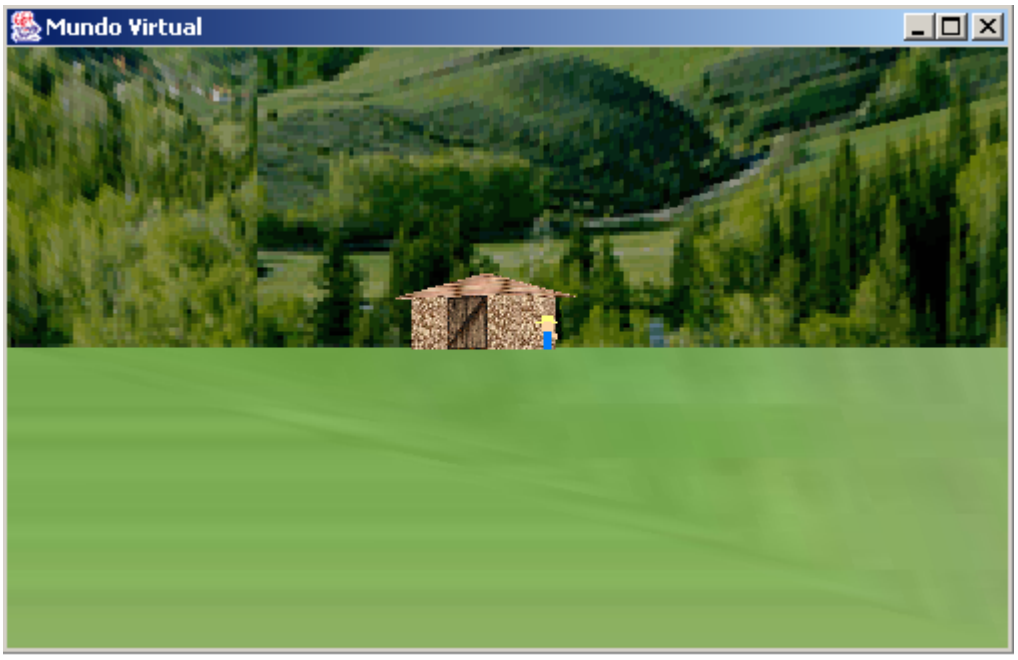


Figura 45 Posición después de acercarse respecto a la posición anterior

- En la figura 46, se muestra cuando se rota la cámara hacia la izquierda con el dedo indicado (anular derecho).

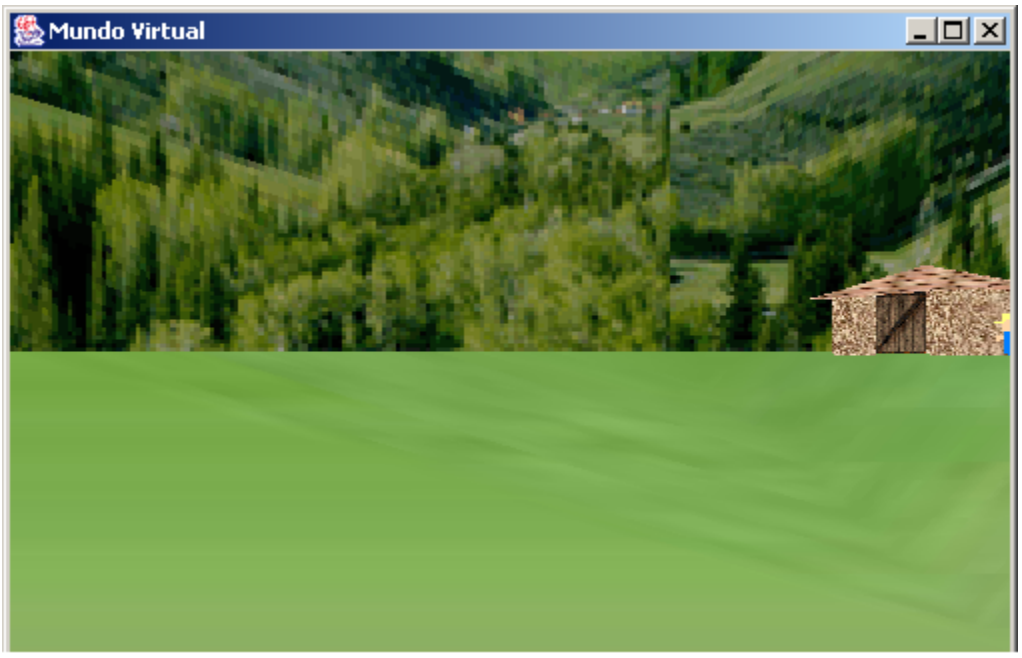


Figura 46 Posición después de rotar dependiendo de la posición anterior

- En la figura 47, es la posición inicial en la que se probo el joystick.

Figura 47. Posición inicial para la prueba del joystick

- En la figura 4, se muestra cuando se aleja la cámara cuando se oprime el eje y, hacia abajo.

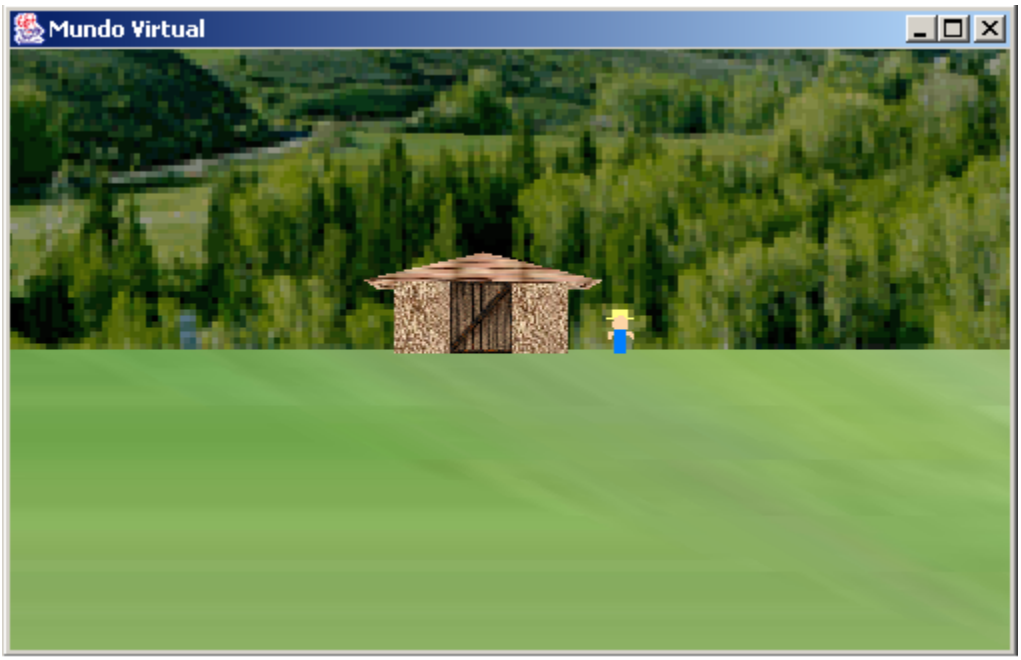


Figura 48 Posición después de alejarse partiendo de la posición anterior

- En la figura 49 se muestra cuando se rota la cámara hacia la derecha cuando se oprime el botón del eje x, hacia la derecha.

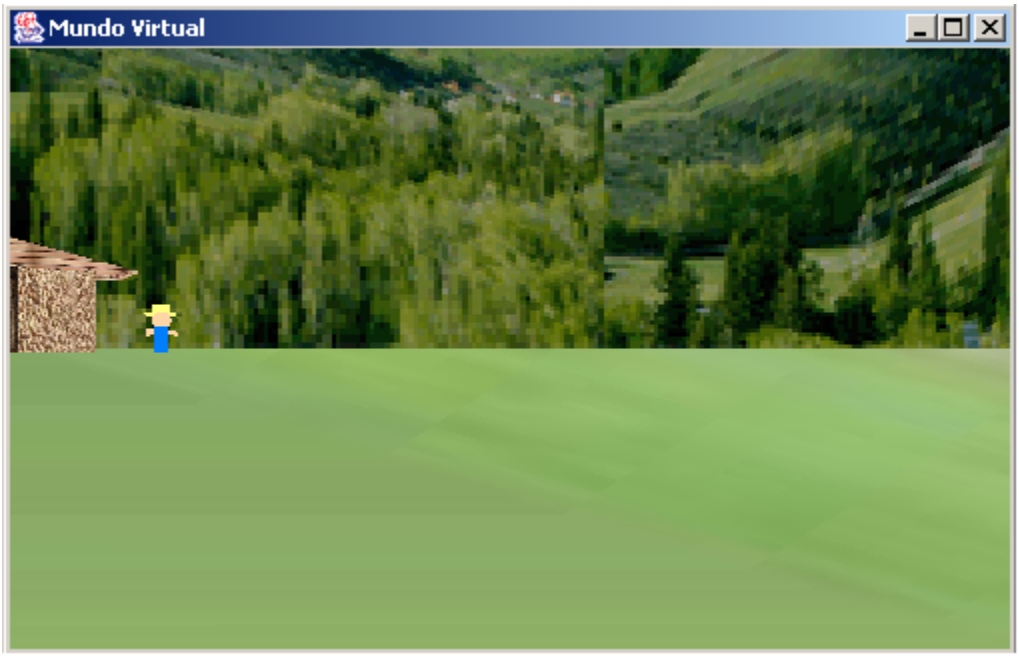


Figura 49. Posición después de rotar a la derecha partiendo de la posición anterior

6.2.2. Segunda prueba

Esta prueba tiene como finalidad, verificar el correcto funcionamiento de la librería al utilizar los tres dispositivos de manera simultanea y concurrente, para poder observar como el mundo virtual se ve afectado de acuerdo a los movimiento de los dispositivos.

Se muestran las imágenes capturadas, con los tres dispositivos conectados, en donde se observa el movimiento de acuerdo a lo descrito en la sección 5.1.3.5.

- En la figura 50, se muestra la posición inicial, antes de empezar a rotar la cámara con el casco y trasladarse con el guante.

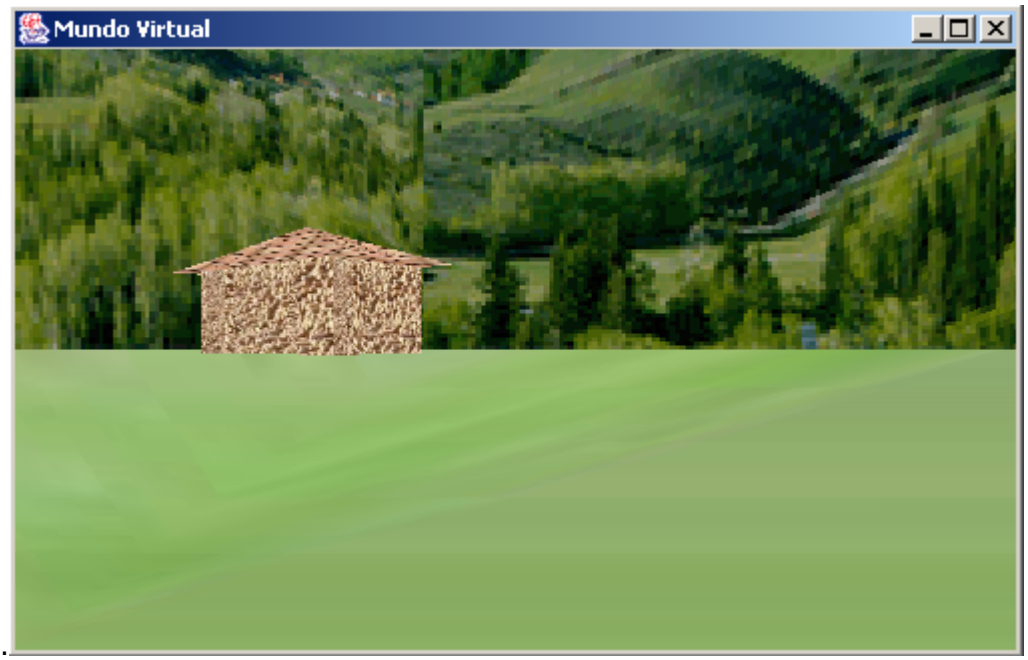


Figura 50 Posición inicial prueba final

- En las figuras 51, 52, 53 y 54 se muestra, la traslación y rotación de la cámara hacia la derecha.

Figura 51. Posición secuencial del movimiento alrededor de la casa I

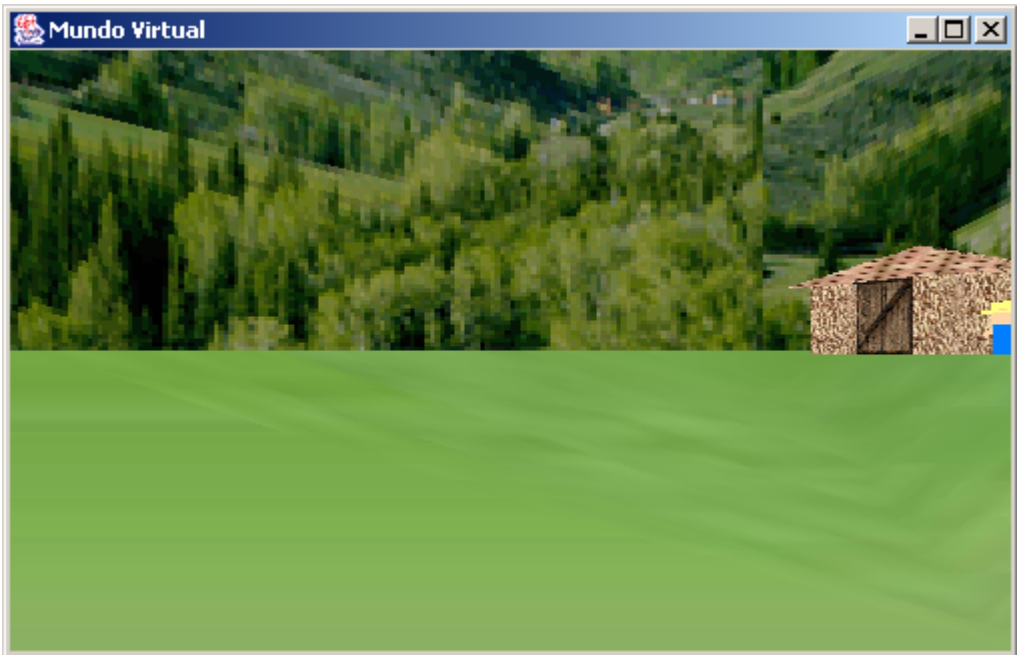


Figura 52 Posición secuencial del movimiento alrededor de la casa II

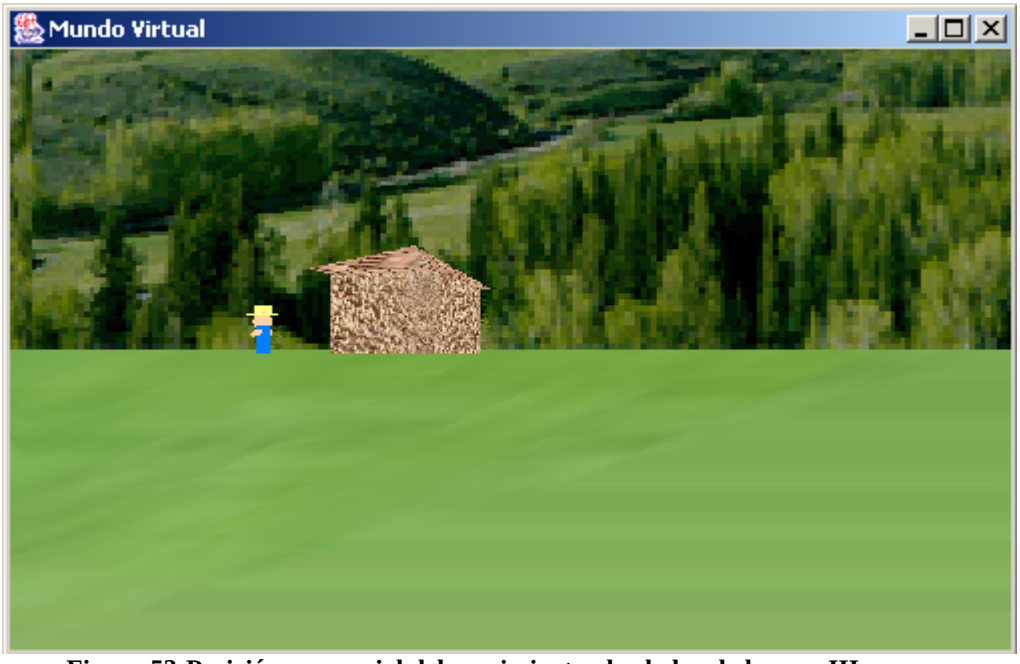


Figura 53 Posición secuencial del movimiento alrededor de la casa III

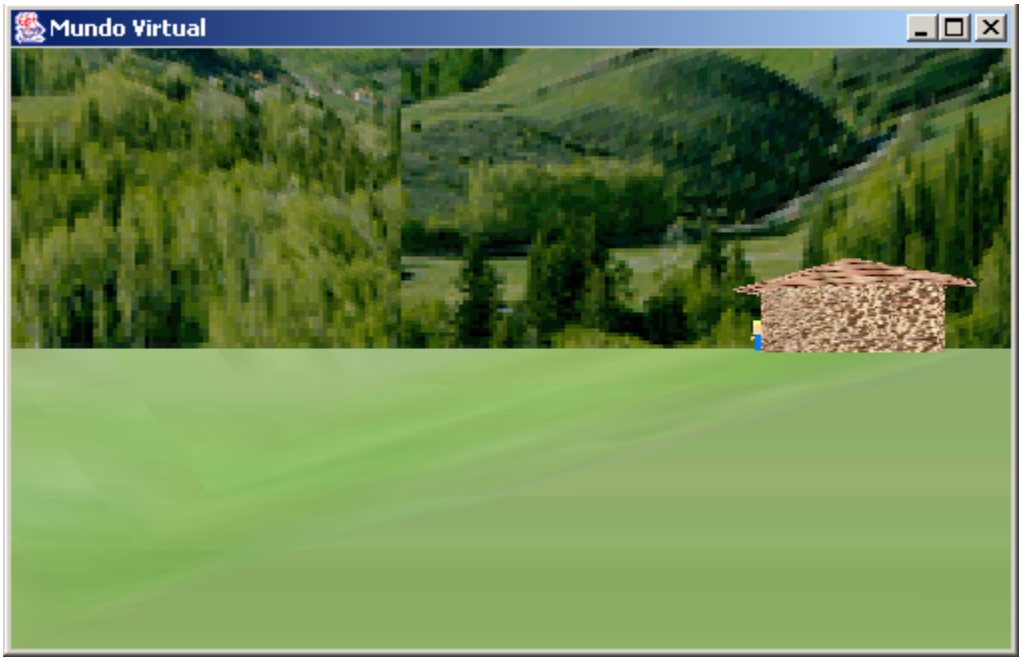


Figura 54 Posición secuencial del movimiento alrededor de la casa IV

6.2.3. Tercera prueba

Esta prueba tiene como objeto, la medición de los tiempos en el procesamiento de los eventos, desde su creación hasta su consumo en la aplicación de realidad virtual, estos tiempos son tomados del sistema para diferentes mundos, los cuales difieren entre si en el número de objetos involucrados en el proceso de renderizado.

Se realizaron pruebas incrementales, estas pruebas consistieron inicialmente en medir los tiempos de administración de eventos con aplicaciones gráficas sencillas que presentaban un reducido número de objetos, posteriormente se utilizaron modelos virtuales más complejos para analizar el desempeño de la librería. En la tabla 7, se observan los tiempos en milisegundos (ms) recogidos durante las pruebas, los cuales representan el tiempo mínimo para procesar un evento generado por un determinado dispositivo.

Vertices	Guante	Casco	Joystick
0	28,83	31,16	31,3
515	28,5	49,33	44,1
1437	26	54,5	91,3
8364	31,16	174,5	210,8
47003	39,33	200,1	279,1

Tabla 7 Pruebas del modulo de eventos

Para entender las pruebas realizadas y dar una mejor interpretación a los datos, se graficaron y se muestran en la figura 55. Los tiempos que se demoran en el procesamiento de los datos desde la creación del evento hasta que se consume de la cola, es incremental, de acuerdo al numero de vértices que se tenga dentro del mundo virtual, las diferencias entre cada uno de los dispositivos se debe a los tiempos que se colocaron para la generación de los eventos, es decir, que se manejaron unos intervalos para reconocer cuando se realiza un evento, de la misma manera, los hilos creados para la lectura de los dispositivos, son diferentes y por consiguiente el dispositivo con un tiempo mas corto responde mas rápido a los eventos, como lo es el guante, los tiempos entre el joystick y casco son iguales por eso tienden a aumentar el tiempo de respuesta desde la cola hasta que se procesan dependiendo del numero de vértices que contenga el mundo.

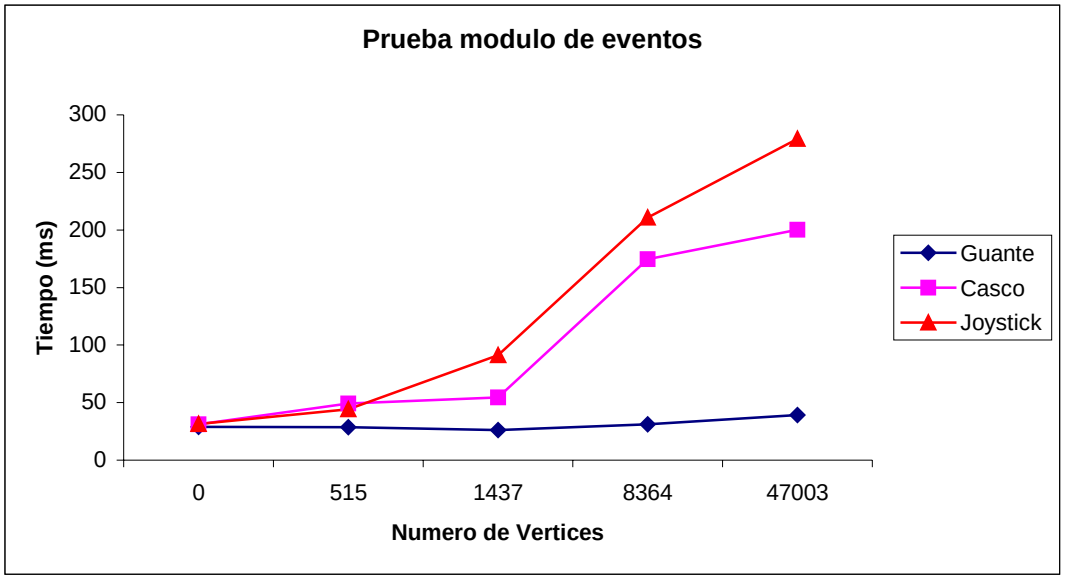


Figura 55 Gráfica de la prueba del módulo de eventos

6.2.4. Cuarta prueba

Esta prueba tiene por objetivo la verificación del funcionamiento de la librería en diferentes sistemas operativos, para corroborar el desempeño de los guantes de datos se utilizó la aplicación de configuración en el sistema operativo Solaris; esta verificación se puede observar en la figuras 56 y 57, en donde se ve la interfaz de la aplicación y donde se observa el guante de datos 5W.

Figura 56 Prueba del guante en solaris

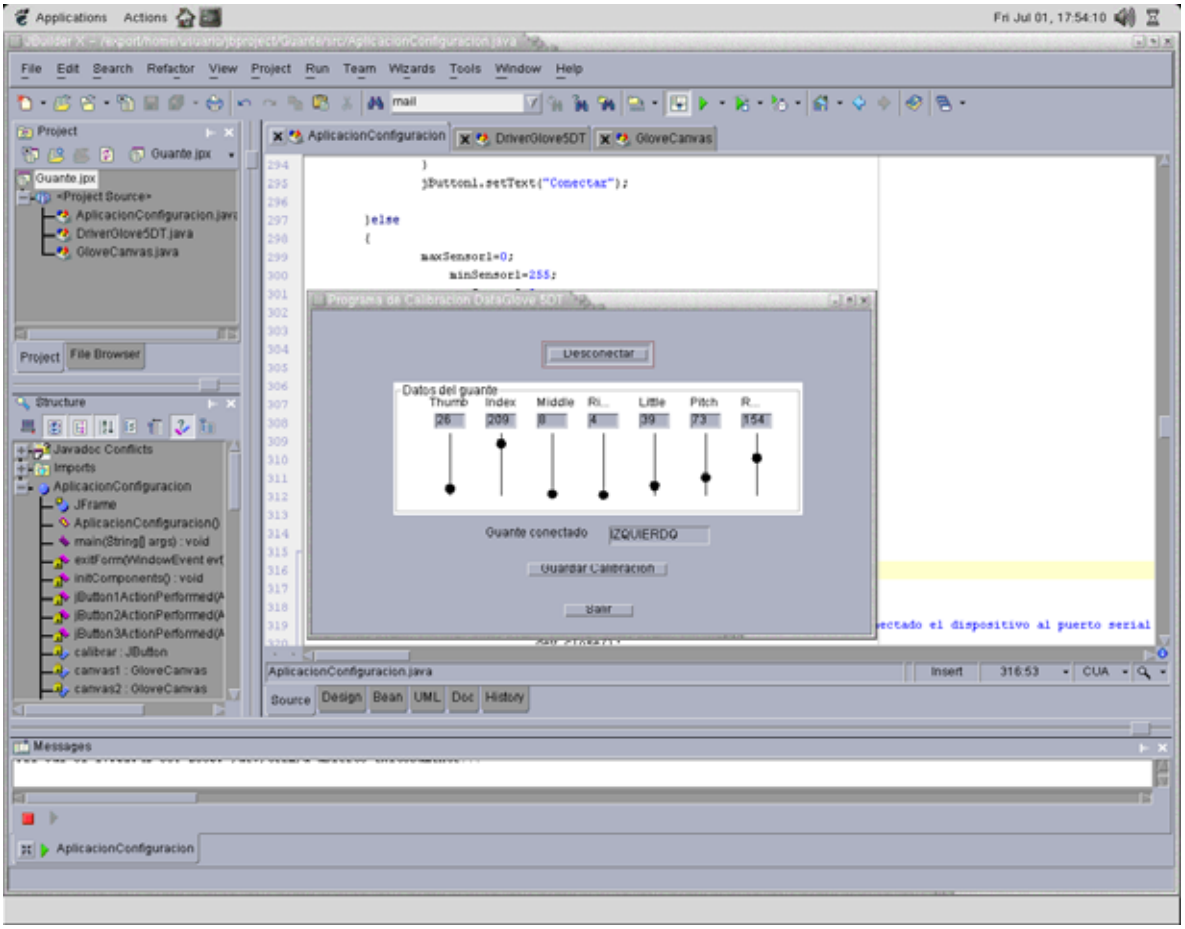


Figura 57 Prueba del guante en Solaris

Para el caso del gamepad, en linux se muestra la aplicación que muestra en consola como varían los datos dependiendo de la acción de los botones o ejes que se estén interactuando, como se ve en la figura 58.

Figura 58 Prueba gamepad en Linux

6.3. Análisis de las pruebas

Con base en las pruebas que se realizaron para los drivers del guante de datos y del casco, se puede concluir que el desempeño que presentan, es semejante al de los drivers que proveen los fabricantes de cada uno de los dispositivos, ya que las variaciones o errores encontrados entre la cantidad de paquetes por segundo que se capturan en cada uno de los drivers no son significativos de acuerdo a los grados de libertad y de la capacidad de envío de los dispositivos físicos.

De acuerdo a las aplicaciones de configuración de los dispositivos, se pudo verificar el correcto funcionamiento tanto del driver como de la librería, ya que éstas permiten validar la implementación frente a los requerimientos inicialmente definidos, como por ejemplo la captura de la información del estado actual de los dispositivos, el reconocimiento del tipo de guante conectado (derecho o izquierdo) o guardar los datos de configuración mínimos para el funcionamiento posterior dentro de la aplicación virtual.

Al realizar las pruebas del joystick, se concluyó que la envolvente (wrapper) implementada, presenta varias ventajas especialmente en el reconocimiento de manera transparente del tipo de dispositivo conectado (gamepad o joystick) y en la captura de la información del estado actual de ejes y botones.

Una vez realizadas las pruebas 1 y 2 se pudo observar que el acoplamiento entre la aplicación y los tres dispositivos empleados funciona correctamente y por la estructura de la librería es transparente la incorporación por parte del programador, puesto que las labores de configuración no son complejas y a nivel de programación no es muy diferente al manejo de eventos que se realiza comúnmente en las aplicaciones de entorno gráfico en Java. El mecanismo de eventos proporcionado permite también asignar de manera sencilla, a nivel de código, cada una de las acciones que se desea realizar con los dispositivos.

Al finalizar la prueba de administración de eventos (prueba 3) se puede concluir que el desempeño de la librería es afectado por la complejidad de las aplicaciones gráficas que la utilicen, es decir, a medida que las aplicaciones presenten un mayor número de objetos para el renderizado, el número de eventos que se pueden procesar por segundo disminuye. Pero esto es normal en las aplicaciones de computación grafica, lo importante aquí, es que la librería no genera el retardo en el proceso de renderizado.

De acuerdo a la prueba 4, se concluye que el núcleo principal de la librería puede emplearse en diferentes sistemas operativos de manera transparente, la principal limitante son los drivers que se utilicen. La portabilidad de uso para los dispositivos que posee la Universidad Javeriana se presenta en la tabla 8, pero existe la posibilidad de ampliar dicha portabilidad especialmente para el casco VFX3D y para los joysticks (ver sección de recomendaciones y trabajos a seguir).

Dispositivo/Sistema Operativo	Windows	Linux	Solaris
Casco VFX3D	SI	NO	NO
DataGlove 5W	SI	NO	SI
Gamepad Logitech	SI	SI	NO
Joystick Logitech	SI	SI	NO

Tabla 8 Portabilidad de la librería para los dispositivos de la PUJ

7. LIBRERÍA DE DISPOSITIVOS DE REALIDAD VIRTUAL

En esta sección se describirá el modo de instalación de la librería, posteriormente se describirá una guía para el desarrollo de aplicaciones con la librería de dispositivos de realidad virtual, la cual lleva por nombre VRDL (*Virtual Reality Device Library*) y finalmente se describirá una guía de expansión de la librería hacia otros tipos o referencias de dispositivos.

7.1. Instalación de VRDL

Para instalar apropiadamente la librería es necesario instalar previamente las librerías auxiliares que requiere VRDL (ver figura 59), en primera instancia se debe instalar Java Communications API y SDL para Java, a continuación se describen los pasos necesarios para realizar dichas tareas.

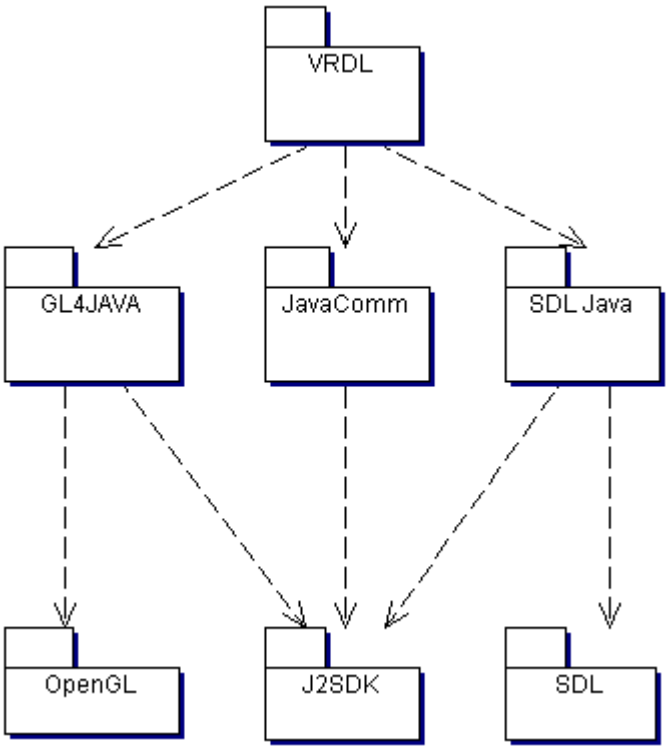


Figura 59 Diagrama de paquetes de VRDL

7.1.1. Instalación de Java Communications API (javacomm)

Esta librería permite administrar principalmente puertos seriales y paralelos, generalmente esta labor es soportada por el sistema operativo, por lo tanto *javacomm* depende del sistema operativo en el que se instale, siendo éste uno de los principales aspectos a tener en cuenta antes de instalar; a continuación se listan los principales requerimientos de la librería:

- Sistema operativo Windows ó Solaris
- Sistema Operativo Linux (utilizando la librería auxiliar RXTX[45])
- JDK 1.3.1 o superior

Esta librería de extensión (versión Windows) consta de los siguientes archivos:

- comm.jar: en este archivo se encuentran el código compilado de la librería
- win32comm.dll: este archivo permite el acceso nativo a los puertos seriales y paralelos.
- javax.comm: este es un archivo de propiedades en el que se encuentra la información necesaria para enumerar los puertos existentes en el equipo.

Pasos de instalación:

- El archivo comm.jar debe ser ubicado en la siguiente ubicación: ***<JAVA_HOME>\jre\lib\ext***
- El archivo win32comm.dll debe ser ubicado en la carpeta system32 y en la siguiente ubicación: ***<JAVA_HOME>\bin***
- El archivo javax.comm debe ser ubicado en la siguiente ubicación: ***<JAVA_HOME>\jre\lib***

7.1.2. Instalación de SDL para Java

SDL para Java es una librería orientada hacia el desarrollo de juegos y basado en la versión en C++ de SDL , por lo tanto es necesario en primera instancia copiar el archivo **SDL.dll** en la carpeta ***system32*** del sistema operativo. En general SDL para Java no tiene grandes restricciones en cuanto al sistema operativo por que existen versiones de esta librería para la mayoría de sistemas operativos conocidos.

SDL para Java esta compuesto por los siguientes archivos:

- sdljava.jar: en este archivo se encuentra el código compilado de la librería
- SDLJava.dll: este archivo administra el manejo de los aspectos gráficos y de acceso a los joysticks.

Pasos de instalación:

- El archivo sdljava.jar debe copiarse en la siguiente ubicación: ***<JAVA_HOME>\jre\lib\ext***
- El archivo SDLJava.dll debe copiarse en la ubicación ***<JAVA_HOME>\bin*** y en la carpeta ***system32***

7.1.3. Instalación de VRDL

Una vez instalados los API auxiliares, de acuerdo a las instrucciones descritas previamente, es posible instalar de manera adecuada la librería de administración y configuración de dispositivos de realidad virtual VRDL, la cual está conformada por los siguientes archivos:

- VRDL.jar: en este archivo se encuentra el código compilado de la librería.
- configuracion.properties: este archivo almacena la configuración de rangos máximos y mínimos para los guantes de datos.
- configuracionHMD.properties: este archivo almacena la configuración de puntos de referencia y de rangos máximos y mínimos para el casco virtual.

Para instalar la librería deben realizarce los siguientes pasos de instalación:

- El archivo VRDL.jar debe copiarse en las siguientes ubicaciones:
  - ***<JAVA_HOME>\jre\lib\ext***
  - ***<JAVA_HOME>\lib***
- Los archivos de configuración deben copiarse en la siguiente ubicación:
  - ***<JAVA_HOME>\jre\lib\ext***

7.2. Desarrollo de Aplicaciones con VRDL

El principal objetivo de la librería de dispositivos de realidad virtual VRDL, es el desarrollo de aplicaciones gráficas Java de tal forma que estas puedan emplear dispositivos de entrada físicos, uno de los aspectos importantes dentro de VRDL es la administración de drivers, la cual permite acoplar diferentes periféricos a la librería. A continuación se describe en primera instancia la estructura de paquetes de VRDL, posteriormente las tareas que debe realizar el desarrollador para

manejar los drivers en VRDL y por último se describen los pasos necesarios para desarrollar aplicaciones de acuerdo a los dos modos de operación que proporciona la librería.

7.2.1. Estructura de paquetes de la librería

Básicamente VRDL esta conformado por tres paquetes los cuales se pueden importan a las aplicaciones Java como se puede apreciar en la figura 60:

```
import edu.puj.VRDL.*;
import edu.puj.VRDL.event.*;
import edu.puj.VRDL.Joystick.*;
```

Figura 60 Paquetes VRDL importados en código Java

El paquete central de la librería se encuentra en **edu.puj.VRDL**, en donde se encuentra la administracion y configuacion de guantes de datos y cascos virtuales; en el paquete **edu.puj.VRDL.event** se encuentran las clases que permiten el manejo de eventos y los *listener* que necesitan las aplicaciones que trabajen en modo de eventos; en el paquete **edu.puj.VRDL.Joystick** se tienen las clases que permiten el descubrimiento y administración de Joysticks.

7.2.2. Manejo de drivers en VRDL

Una de las tareas más importantes que debe realizar el desarrollador de aplicaciones gráficas con VRDL, es cargar los drivers a la librería, para hacer esto es necesario que los drivers se encuentren en el CLASSPATH del ambiente de ejecución Java, sólo es necesario incluir la ruta en la que se encuentra el driver dentro de la variable de entorno CLASS_PATH. Posteriormente el desarrollador debe crear una instancia del driver y pasarla como parámetro a la librería como se puede apreciar en la figura 61.

```
Class driver=Class.forName("DriverHMDVFX3D");
Device casco=Device.getDevice("HMD",driver.newInstance());
```

Figura 61 Manejo de drivers en Java

Es importante tener en cuenta que los drivers que deber ser cargados en VRDL deben ser implementaciones de las interfaces **DriverGlove** y **DriverHMD,** las cuales se encuentran el paquete **edu.puj.VRDL.**

7.2.3. Modos de operación de VRDL

Para establecer la comunicación con los dispositivos, VRDL establece dos modos de operación diferentes los cuales deben ser utilizados de acuerdo a las necesidades del desarrollador. A continuación se describen dichos modos de operación y como deben ser empleados en las aplicaciones Java.

7.2.3.1. Request Mode

En este modo de operación la aplicación solicita directamente a la librería los datos del estado actual del dispositivo, por lo tanto el desarrollador debe realizar algunas tareas como asegurar que la aplicación obtenga la información del dispositivo en un hilo de procesamiento, para no bloquear el flujo principal de ejecución de la aplicación.

En primera instancia, el desarrollador debe crear la instancia del dispositivo lógico, para posteriormente abrir la comunicación en un modo de operación determinado en el cual trabajará dicha instancia, en la figura 62 se muestra la manera de realizarlo en la aplicación.

```
Class driver=Class.forName("DriverHMDVFX3D");
Device casco=Device.getDevice("HMD",driver.newInstance());
Boolean estado =casco.open("COM1",HMD.REQUEST_MODE);
```

Figura 62 Inicialización de dispositivos lógicos en VRDL

Para trabajar con joysticks el proceso de inicialización es diferente, como se puede apreciar en la figura 63, ya que el administrador de Joysticks enumera los dispositivos desde 0 hasta el numero de periféricos conectados menos 1.

```
try {
JoystickManager manager=new JoystickManager();
int numjoys=manager.getNumJoystick();
if(numjoys>=1){
  for(int i=0;i<numjoys;i++){
   Joystick j=manager.openJoystick(i,JoystickManager.REQUEST_MODE);
  }
}
}catch (Exception ex) {
   ex.printStackTrace();
}
```

Figura 63 Inicialización de joysticks en VRDL

Una vez realizado esto, se puede solicitar la información del estado actual del dispositivo , como se muestra en la figura 64.

```
Thread t=new Thread()
{
      public void run()
      {
         while(leer)
         {
            int[] res=casco.getSensorsData();
          //REALIZAR LO QUE REQUIERA LA APLICACIÓN CON LA
          //INFORMACION DE LOS SENSORES
         }
      }
};
t.start();
```

Figura 64 Obtención de la información en Request Mode

Para obtener la información del actado actual de los joystick se debe solicitar la información como se muestra en la figura 65.

```
int numaxis=j.getNumAxis();
int numbotones=j.getNumButtons();
Thread t=new Thread()
{
    public void run(){
      while(true){
         for(int k=0;k<numaxis;k++)
         {
             if(j.getAxis(k)!=0 && j.getAxis(k)!=-1)
             {
               System.out.print("Eje "+k+": "+j.getAxis(k)+" ");
               //Procesar lo que la aplicación requiera con la
               // informacion de ejes
             }
         }
         for(int k=0;k<numbotones;k++){
            if((j.getButton(k)).isPressed()){
               System.out.println("Boton "+k+" presionado!!!!");
              //Procesar lo que la aplicación requiera con la
              //información del botón
          }
      }
   }
}
t.start();
```

Figura 65 Obtención de la información de joysticks en Request Mode

7.2.3.2. Event Mode

Este modo de operación permite que la aplicación sea notificada por el manejador de eventos de VRDL, de tal forma que la librería realiza muchas de las tareas que el desarrollador debe hacer en Request Mode.

Para emplear este modo de operación lo primero es implementar uno de los *listener* que se encuentran en el paquete edu.puj.VRDL.event, de acuerdo al dispositivo con el que se vaya a trabajar, como se muestra en la figura 66.

```
import edu.puj.VRDL.event.GloveListener;
import edu.puj.VRDL.*;
import edu.puj.VRDL.*;
public class TestGlove extends JFrame implements GloveListener{
    public void gloveMoved(){
       //Procesar la información del estado actual del dispositivo
       //la cual se encuentra en el evento
    }
}
```

Figura 66 Implementación de la interfaz GloveListener

La inicialización del dispositivo en Event Mode se realiza de manera análoga a la inicialización en request Mode, como se puede apreciar en la figura 67.

```
Class driver=Class.forName("DriverGlove5DT");
Device guante=Device.getDevice("DataGlove",driver.newInstance());
Boolean estado =guante.open("COM1",DataGlove.EVENT_MODE);
```

Figura 67 Inicialización de dispositivos en Event Mode

Una vez se establece la comunicación con el dispositivo lógico, este empieza a generar eventos a medida que el estado actual del periférico cambie, pero el manejador de eventos necesita tener registrados los *listener* a los cuales va a notificar, por lo tanto es necesario registrar el *listener* implementado ante el *EventManager*, como se puede apreciar en la figura 68, una vez realizado este registro la aplicación será notificada de cada un de los evento que se generen.

```
EventManager man=EventManager.getInstance(50);
man.addGloveListener(this);
```

Figura 68 Registro de listeners ante el manejador de eventos de VRDL

El parámetro de la función **getInstance(long timeout),** especifica el tiempo que el manejador de eventos espera por la llegada de un nuevo evento, para mas información sobre como emplear la librería VRDL se incluyen los ***javadocs*** con los archivos fuente de la librería.
Para tener una idea mucho más clara del modo de uso de VRDL se incluye el diagrama de actividades de la figura 69.

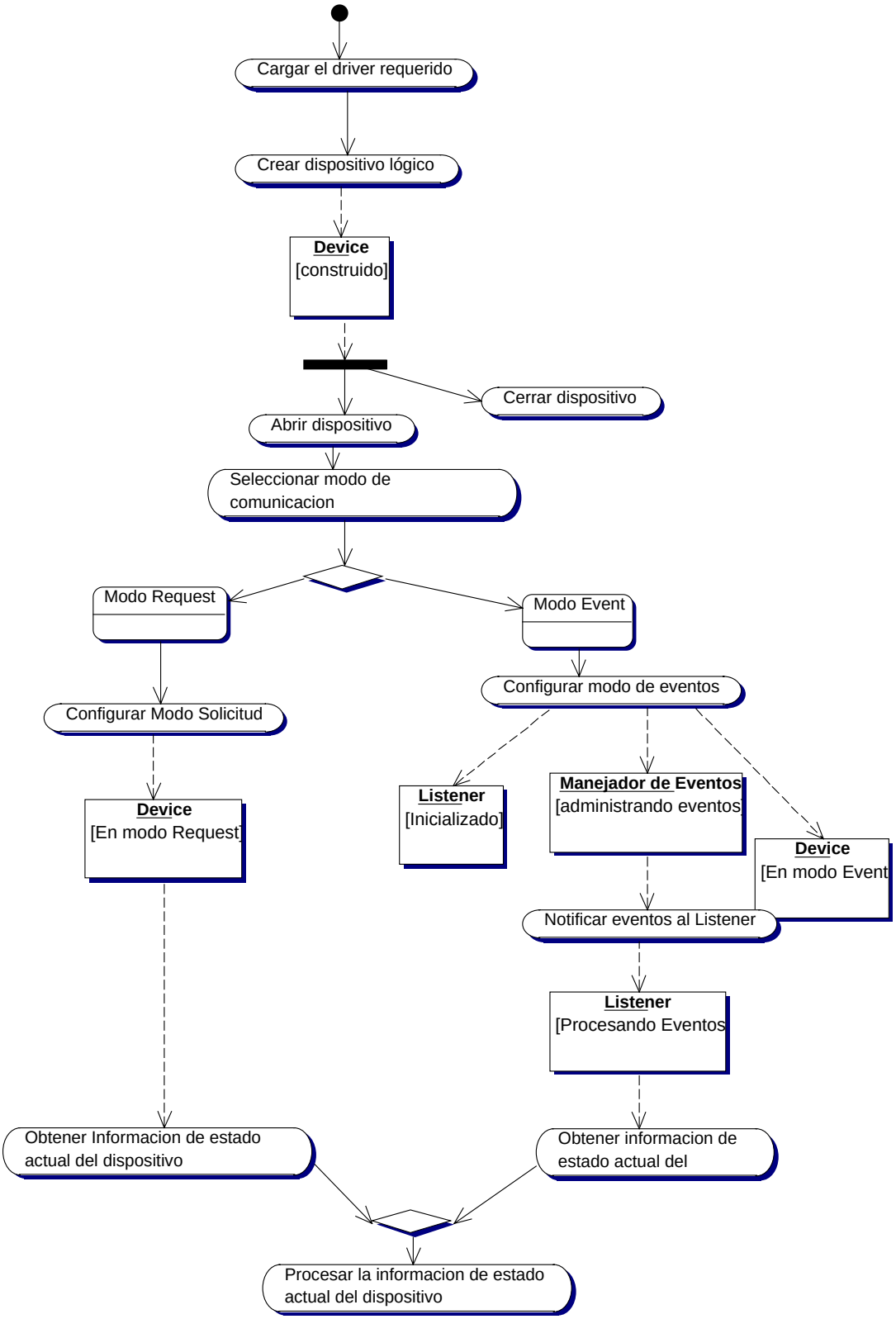


Figura 69 Diagrama de actividades de la librería

Para entender mejor la administración de joysticks, se puede apreciar el diagrama de actividades de la figura 70.

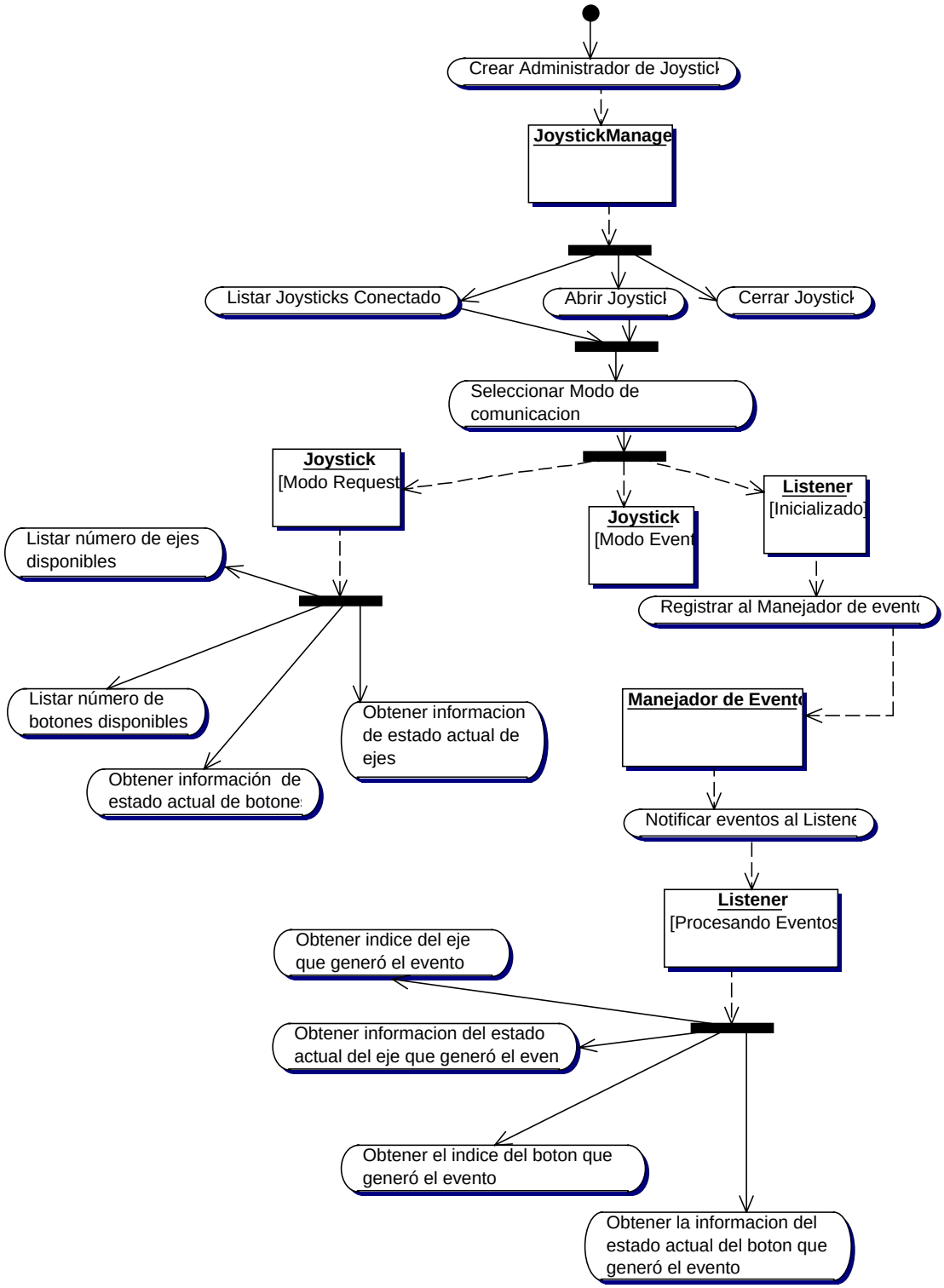


Figura 70 Diagrama de actividades para administrar joysticks

7.2.3.3. Cuando debe ser utilizado un determinado modo de operación

El hecho de utilizar un determinado modo de operación puede afectar en gran manera el desempeño de las aplicaciones, dependiendo de las características propias de la aplicación se debe escoger entre uno u otro modo, a continuación se explica cuando deben ser empleados los modos Request y Event.

- Request Mode: se recomienda para aplicaciones livianas que requieran un grado alto de precisión en la información del estado actual de la información, por ejemplo las aplicaciones de configuración de VRDL están implementadas en este modo de operación, debido a la importancia que debe tener la precisión y la exactitud de este proceso.

- Event Mode: se recomienda especialmente para las aplicaciones gráficas y de realidad virtual, ya que permite que labores como el renderizado no se ejecuten en todo momento, sino solamente cuando se genera un evento en los dispositivos.

7.3. Expansión de la librería

En general la librería puede expandirse para poder adaptar otros tipos de dispositivos u otras referencias de periféricos, pero hay que tener en cuenta ciertos detalles importantes a la hora de realizar una expansión de la librería, en esta sección se explicarán dichos aspectos.

7.3.1. Cómo utilizar otras referencias

Si el objetivo es extender la librería hacia otras referencias de dispositivos diferentes al casco **VFX3D** y al guante de datos **5W**, se deben tener en cuenta las siguientes restricciones:

- El tipo de dispositivos debe ser basado en sensores de movimiento.
- Solamente se pueden adaptar dispositivos del tipo de guantes de datos y cascos.

Para poder adaptar otro tipo de referencias deben seguirse los siguientes pasos:

- Analizar la estructura de las librerías HMD y DataGlove.
- Determinar bajo que librería se adapta mejor el dispositivo a incluir dentro de la librería.

- Construir el driver asociado al dispositivo de acuerdo a la estructura definida por las interfaces **DriverGlove** ó **DriverHMD,** es decir el driver que se desee construir debe implementar los métodos definidos por alguna de las interfaces antes mencionadas.
- Para la construcción del driver se deben tener en cuenta los siguientes aspectos:
    - El driver es una clase Java que debe implementar las interfaces definidas para guantes de datos y cascos.
    - El driver debe permitir como mínimo el establecimiento de la comunicación con el dispositivo físico, enumerar los sensores que posea el periférico, y obtener el estado actual en valores numéricos de los sensores. De ser posible se sugiere que dichos valores numéricos estén entre los rangos que manejan los dispositivos con los que trabaja la librería inicialmente, estos son:
        - Para el casco los valores de los sensores están entre 0 y 65536.
        - Para el guante de datos cada sensor tiene un valor numérico entre 0 y 255.
    - El driver debe utilizar algún mecanismo Java para acceso al dispositivo físico (ver marco teórico), así como los parámetros de configuración de puertos (si es necesario) y los protocolos de comunicación.
- Una vez construido el driver del nuevo dispositivo este debe incluirse en el classpath del sistema para que este pueda ser cargado en memoria mediante la instrucción **Class.forName(String nombreDriver),** esta instrucción retorna un objeto del tipo **Class,** luego debe crearse una instancia del driver mediante la instrucción **newInstance()** y pasarle la instancia a la librería con el tipo de dispositivo correspondiente.

7.3.2. Como emplear diferentes tipos de dispositivos

Si se desea emplear dentro de la librería otros tipos de dispositivos diferentes a guantes de datos y cascos para realidad virtual, se debe tener en cuenta que el dispositivo a adaptar debe estar constituido por sensores de movimiento, para poderlo adaptar se deben seguir los siguientes pasos:

- Se debe construir una clase que herede de **Device** e implemente sus métodos, dichos métodos deben cumplir la funcionalidad básica del dispositivo basado en sensores.
- Se debe definir en el método **open** la posibilidad de seleccionar el modo de comunicación por solicitud de información ó por notificación de eventos, para implementar dichos modos de comunicación se debe tener en cuenta lo siguiente:

    - En el modo **Request** el usuario puede acceder directamente a la información de estado actual del dispositivo mediante el llamado al método **getSensorsData.**
    - En el modo **Event** se debe implementar un hilo que este consultando periódicamente el dispositivo y al detectar un cambio en el estado actual des dispositivo se debe crear una instancia de **DeviceEvent** con la información de los sensores, para posteriormente colocarlo en la cola de eventos de la librería (**EventReportQueue**), en este modo debe inhabilitarse el uso del método **getSensorsData**.
- Se debe definir una interfaz que defina el comportamiento mínimo de los drivers para el tipo de dispositivo a desarrollar y se debe agregar un método de construcción de driver en la interfaz **DriverFactory** y en la clase **ConcreteDriverFactory**, en esta última se debe realizar el proceso de *casting* de la instancia que le entra a la librería para poder retornar el tipo de driver al que pertenece.

8. CONCLUSIONES

- La librería para el manejo y configuración de dispositivos de realidad virtual, implementada en la investigación permite emplear periféricos tales como cascos, guantes de datos y joysticks en aplicaciones gráficas y entornos virtuales desarrollados en Java de manera sencilla y acoplada.

- Las herramientas de configuración facilitan el trabajo del desarrollador de aplicaciones gráficas y de realidad virtual, para establecer parámetros de ejecución dependiendo de las necesidades de la aplicación a desarrollar y de las características del entorno físico.

- La librería funciona de manera adecuada en equipos que cumplen con los requisitos mínimos especificados en este documento, en otro tipo de equipos con recursos de hardware insuficientes se recomienda no usar más de un dispositivo a la vez, esto para no agotar los recursos del sistema y evitar caídas del mismo.

- El núcleo principal de la librería desarrollada es portable a sistemas operativos Windows. Linux y Solaris de manera transparente para su uso desde aplicaciones Java.

- La portabilidad de la librería solamente puede verse limitada por la portabilidad de los drivers que se desarrollen para la misma, en el caso de los drivers desarrollados en esta investigación para los dispositivos que posee la Universidad Javeriana, se tiene que para el guante de datos la portabilidad se limita a los sistemas operativos que soporten **Java Communications API** (Windows, Solaris, Linux)**,** para los joysticks la portabilidad esta dada por SDLJAVA la cual se encuentra desarrollada para la sistemas operativos Windows y Linux, mientras que para el casco virtual se tiene que el driver desarrollado funciona bajo sistemas operativos Windows únicamente, esto debido al manejo de archivos DLL, el cual no puede ser empleado en otros sistemas operativos, para los cuales debe generarse el archivo que realice la funcionalidad correspondiente.

- El desempeño de la librería desarrollada, varía de acuerdo a la complejidad de las aplicaciones gráficas que la utilicen, de acuerdo al número de objetos que se encuentren asociados al proceso de renderizado de dichas aplicaciones.

- El documento resultado de esta investigación proporciona un modelo de administración y configuración de dispositivos de realidad virtual, basado en

la investigación de modelos conceptuales previos, el cual es un aporte para el desarrollo de aplicaciones de realidad virtual, debido al número reducido de aplicaciones y modelos de código abierto en cuanto a la administración y configuración de este tipo de dispositivos.

- OpenGL para Java es una buena herramienta para el desarrollo de aplicaciones gráficas y de realidad virtual, puesto que es una implementación basada en el OpenGL original, por lo tanto proporciona toda la funcionalidad necesaria para el desarrollo de aplicaciones livianas y robustas.

- OpenGL para Java es una librería gráfica cuya versión no se actualiza desde el año 2002, por lo tanto se recomienda emplear librerías más actualizadas como JOGL y Java3D. El funcionamiento de la librería desarrollada en esta investigación no cambiará puesto que su estructura no depende de las librerías gráficas.

- El desarrollo y análisis de las pruebas confirma que se pueden acoplar tres diferentes tipos de dispositivos físicos con una aplicación gráfica desarrollada en Java de manera ágil y sencilla, lo cual responde a la pregunta inicial que motivó la realización de la investigación, la cual establecía el interrogante sobre la posibilidad de desarrollar aplicaciones Java para realidad virtual con manejo de dispositivos físicos de interacción.

- El desarrollo de la librería de administración de dispositivos de entrada para realidad virtual, permitió emplear algunos de los dispositivos que posee la Universidad Javeriana, como el guante de datos, en sistemas operativos en los cuales no se habían empleado, como Solaris, puesto que los drivers para este sistema operativo son costosos, lo cual es un pequeño paso hacia la expansión de la realidad virtual y de la computación grafica a otros sistemas operativos.

- La implementación de la librería, resultado de la presente investigación, no pretende ser la única forma de administrar y configurar dispositivos de entrada para realidad virtual, pero si pretende ser la base de muchas otras investigaciones en este campo, que permitan manejar dispositivos mucho más complejos que existen en el momento y que irán apareciendo con el paso del tiempo.

9. RECOMENDACIONES Y TRABAJOS A SEGUIR

En esta sección se presentarán las principales recomendaciones que se sugieren seguir para el mejoramiento de la librería desarrollada y algunos proyectos futuros que pueden surgir con base en esta investigación.

RECOMENDACIONES

- Para ampliar el grado de portabilidad de la librería, se sugiere construir el driver nativo para el funcionamiento del casco VFX3D en otros sistemas operativos diferentes a Windows.

- Para extender la utilidad de la librería desarrollada se recomienda implementar el manejo de **Force Feedback** dentro para dispositivos como joysticks y guantes da datos.

- Actualmente la librería **SDLJAVA** se encuentra disponible para sistemas operativos Windows y Linux únicamente, en el sitio oficial del proyecto SDLJAVA**[41]** se encuentra el código fuente de dicha librería; se recomienda realizar las versiones de **SDLJAVA** para otros sistemas operativos.

TRABAJOS A SEGUIR

- Acoplar la librería desarrollada con librerías gráficas como JOGL o OpenGL for Java, para realizar el desarrollo de aplicaciones para realidad virtual en lenguaje Java de una manera mucho más sencilla.

- Extender la librería hacia otros tipos de dispositivos para realidad virtual mas complejos estructuralmente como trackers o CAVEs.

10. BIBLIOGRAFÍA

[1] Java Communications API (JCA), http://java.sun.com/products/javacomm/, última visita: 13 de Marzo de 2005.

[2] Java Native Interface (JNI), http://java.sun.com/j2se/1.3/docs/guide/jni/, última visita: 1 de Junio de 2005.

[3] Java3D, http://www.j3d.org/, última visita: 28 de Noviembre de 2004.

[4] Java Technology, http://java.sun.com/, última visita: 1 de Junio de 2005

[5] University of Oulu, http://www.oulu.fi/, última visita: 28 de Noviembre de 2004.

[6] DigiLoop, http://www.hci.oulu.fi/digiloop/, última visita: 5 de Octubre de 2004.

[7] http://www.cs.jhu.edu/~cohen/VW2000/Lectures/History.bw.pdf. ,última visita: 18 de Septiembre de 2004

[8] http://www.monografias.com/realidadvirtual.htm última visita: 18 de Septiembre de 2004

[9] VR Juggler: A Framework for Virtual Reality Development, http://www.vrjuggler.org/pub/vr.juggler.framework.for.vr.dev.1998.IPT98.pdf, última visita: 18 de Septiembre de 2004

[10] http://usuarios.lycos.es/artofmusic/the_matrix_vr/historia_vr.html

[11] Avocado: A Distributed Virtual Environment Framework, Henrik Tramberend, Marzo 2003, última visita: 6 de Octubre de 2004

[12] Opengl, http://www.opengl.org, última visita: 1 de Junio de 2005.

[13] DirectX, http://www.microsoft.com/windows/directx/default.aspx, última visita: 22 de Septiembre de 2004.

[14] Instituto Tecnológico y de Estudios Superiores de Occidente, http://148.201.1.19/virtual/01a7.html , última visita: 25 de Agosto de 2004.

[15] Universidad tecnológica de Panamá, http://www.utp.ac.pa/seccion/topicos/realidad/, última visita: 6 de Noviembre de 2004.

[16] Welcome to my World, http://dbarrero.tripod.com, última visita: 6 de Noviembre de 2004.

[17] ACM crossroads student magazine, http://www.shef.ac.uk/~vrmbg/, última visita: 6 de Noviembre de 2004.

[18] Laboratorio Docente de Computación, http://www.ldc.usb.ve/~97-29640/Estereoscopia.htm, última visita: 6 de Noviembre de 2004.

[19] http://psych.hanover.edu/Krantz/MotionParallax/MotionParallax.html, última visita: 6 de Noviembre de 2004.

[20] Manual de Java, http://www.geocities.com/CollegePark/Quad/8901/cap08.htm. ,última visita: 14 de Septiembre de 2004.

[21] Java3D Internal Architecture, http://www.j3d.org/implementation/architecture.html, última visita: 14 de Septiembre de 2004.

[22] Java 3D API, http://java.sun.com/products/java-media/3D ,última visita: 20 de Noviembre de 2004.
[23] A First Step Towards Object-Oriented Architectures for Virtual-Reality Applications, http://citeseer.nj.nec.com/context/171622/ ,última visita: 25 de Agosto de 2004.
[24] Mesa 3D, http://www.mesa3d.org ,última visita: 25 de Agosto de 2004.
[25] OpenGL for JAVA, www.jausoft.com/gl4java/ ,última visita: 1 de Junio de 2005.
[26] Baecker, R.M., Buxton W. (1987), *Readings in human-computer interaction : A multidisciplinary approach*, 357-365.
[27] Card, Stuart K., Mackinlay Jock D., Robertson, George G. (1990), *The design space of input devices* 117-124.
[28] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F. (1990), *Computer Graphics : Principles and Practices*, Addison-Wesley
[29] Langolf, G.D. (1973), *Human Motor Performance in Precise Microscopic Work*, University of Michigan.
[30] ISO, *"Information Processing Systems – Computer Graphics – Graphical Kernel System (GKS)"*, ISO 7942 (1985).
[31] www.iso.org, última visita: 5 Mayo de 2005.
[32] Duce, D. A., van Liere,R., ten Hagen, P.J.W. *"Components, Frameworks and GKS Input"* (1989) 92-94, Rutherford Appleton Laboratory (UK), CWI (Netherlands).
[33] Protocolo USB, www.usuarios.lycos.es/kurganz/introduccion.html, última visita: 26 Mayo de 2005.
[34] JOGL, www.genedavissoftware.com/books/jogl/ljogl_ch1.html 2005-05-23, última visita: 10 Mayo de 2005.
[35] Seagal, Mark y Akeley Kurt, *The OpenGL Graphics System: A Specification* (Version 2.0 - October 22, 2004).
[36] Bruegge, Bernd, *Ingeniería de software orientada a objetos*, Prentice Hall (2002).
[37] Fifth Dimension Technologies , www.5dt.com , última visita: 1 de Junio de 2005.
[38] Interactive Imaging Systems, www.iisvr.com , última visita: 21 de Abril de 2005.
[39] Joystick Controllers Specifications, http://www.globalspec.com/specifications/spechelpall?name=Joystick_Controllers&comp=1399 , última visita: 10 de Mayo de 2005.
[40] JUSB, http://jusb.sourceforge.net/ ,última visita: 15 de Abril de 2005.
[41] SDL for Java, http://sourceforge.net/projects/sdljava/ , última visita: 28 de Mayo de 2005.
[42] Eckel, Bruce; Thinking in Patterns (2003).
[43] HHD Software, www.hhdsoftware.com/sermon.html , última visita: 31 de Mayo de 2005.
[44] Serial port monitor software - serial port sniffer & analyzer http://www.kmint21.com/serial-port-monitor/, última visita: 31 de Mayo de 2005.

[45] RXTX serial and parallel I/O libraries supporting Sun CommAPI, www.rxtx.org, última visita: 1 de Junio de 2005.
[46] Paralax Multimedia, http://www.paralax.com.mx/09b_tecnicas.html , última visita: 5 de Julio de 2005.
[47] Razón y palabra, revista electrónica especializada en tópicos de comunicación, http://www.razonypalabra.org.mx/anteriores/n22/22_sfragoso.html, última visita: 5 de Julio de 2005.
[48] Molina Massó, J. P., Tema 2:*Introducción a la realidad virtual*, Escuela Superior de Albacete Páginas 1-60.
[49] Componentes de la estereoscopía en Realidad Virtual, resumbrae.com/talks/ vassar/page03.html, última visita: 5 de Julio de 2005.
[50] Bayona B., Sofía, Realidad Virtual y Animación: Dispositivos de entrada Universidad Rey Juan Carlos,
[51] "Cine con texturas, olor y sabor", artículo periodístico, http://www.celuloide.com.ar/archives/2005/04/09/cine-con-texturas-olor-y-sabor/, última visita: 5 de Julio de 2005.
[52] Cyberworld, http://www.cwonline.com/store/view_product.asp?Product=1147, última visita: 5 de Julio de 2005.
[53] Ware,Colin; Balakrishnan, Ravin , *Reaching for Objectsin VR Displays: Lag and Frame Rate* (2000), University of New Brunswick.

11. ANEXOS

A continuación se muestran los anexos, que se hacen referencia dentro de la investigación.

ANEXO A. PROTOCOLO USB(Universal Serial Bus)

USB es una interfaz para la transmisión serie de datos y distribución de energía desarrollado por el sector de las telecomunicaciones y de los ordenadores **[33]**, para mejorar la velocidad en las interfaces seriales y paralelas.

La interfaz tiene 4 hilos que distribuyen 5 V (voltios) para alimentar el dispositivo y transmitir los datos a una velocidad cercana a 480 Mbps en la versión 2.0. Presenta varias ventajas como la detección y configuración automáticas, debido a que se encuentra conectado a la red y sin ningún software adicional, ni reiniciar el PC.

El USB presenta varios aspectos importante como la topología, el flujo de datos, el protocolo, la parte eléctrica y la mecánica, donde se puede apreciar ventajas en cada una, como se ve a continuación.

La topología se puede explicar como una de estrellas apiladas, donde cada uno de los dispositivos se conecta al host y el hub central o raíz, es el centro de cada estrella.

Los dispositivos de USB, se puede entender como un conjunto de endpoints, donde cada uno de ellos se agrupan en conjuntos que dan como resultado a interfaces, las cuales permiten manejar el dispositivo.

La comunicación es dependiente del nivel en que se encuentre, a nivel mas bajo el host se comunica por medio del cable y la interfaz, a medida que sube el nivel el software se comunica con el dispositivo lógico utilizando la tubería de control por defecto.

Los endpoints son numero asignados a cada, que vienen de fabrica, por eso es único y se caracteriza por la frecuencia de acceso, el ancho de banda, el numero de endpoint, el tratamiento de los errores, máximo tamaño de paquete, tipo de transferencia y la orientación de la misma.

Las tuberías permite controlar el acceso y la comunicación de las transferencias, de acuerdo a dos endpoints que contienen un 0 o 1 en las entradas y salidas, de acuerdo a esto se pueden especificar la tubería ya sea Stream o Mensaje **[33]**.

Stream: Los datos se mueven a través de la tubería sin una estructura definida
Mensaje: Los datos se mueven a través de la tubería utilizando una estructura USB

Existen 4 tipos de transferencias, isócronas, de control, de interrupción y de bultos (bulk), cada una de estas presentan varias características propias, por ejemplo la

de control es la única que maneja tuberías de mensajes, las demás stream, varían de acuerdo a los tamaños de los paquetes y de la información como se maneja..

Figura 71 Protocolo USB

ANEXO B DIAGRAMAS DE CLASE

B1. Casco

Tipo PID	Nombre PID	PID	Descripción
Token	OUT IN SOF SETUP	0001B 1001B 0101B 1101B	Dirección + número de endpoint en una transacción host a función. Dirección + número de endpoint en una transacción función a host. Indicador de inicio de frame (Start Of Frame) y número de frame. Dirección + número de endpoint en una transacción host a función para realizar un Setup de una tubería de control.
Data	DATA0 DATA1 DATA2 MDATA	0011B 1011B 0111B 1111B	PID de paquete de datos par. PID de paquete de datos impar. PID de paquete de datos de alta velocidad, elevado ancho de banda en una transferencia isócrona en un microframe. PID de paquete de datos de alta velocidad para split y elevado ancho de banda en una transferencia isócrona.
Handshake	ACK NAK STALL NYET	0010B 1010B 1110B 0110B	El receptor acepta el paquete de datos libre de errores. El dispositivo receptor no puede aceptar los datos o el dispositivo emisor no puede enviar los datos. Endpoint sin servicio o una petición de control sobre una tubería no está soportado. Aún no se ha recibido una respuesta del receptor.
Special	PRE ERR SPLIT PING Reservado	1100B 1100B 1000B 0100B 0000B	(Token) Habilita trafico de bajada por el bus a dispositivos de velocidad baja. (Handshake) Error de transferencia Split. (Token) Transferencia de alta velocidad Split. (Token) Control de flujo sobre endpoints de tipo control o bulk. PID reservado.

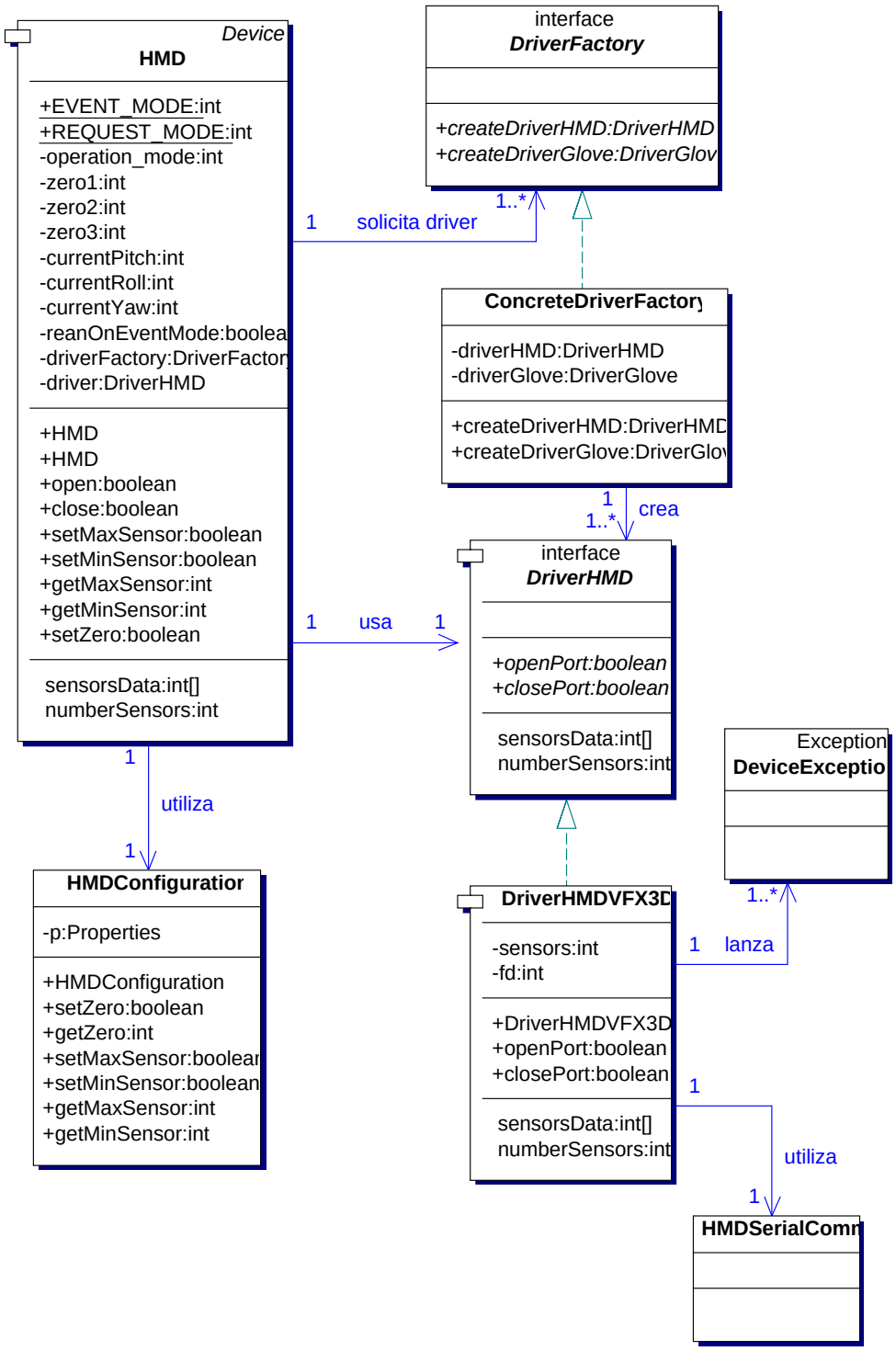


Figura 72 Diagrama de clases a nivel de implementación de la librería para cascos

B2. Joystick

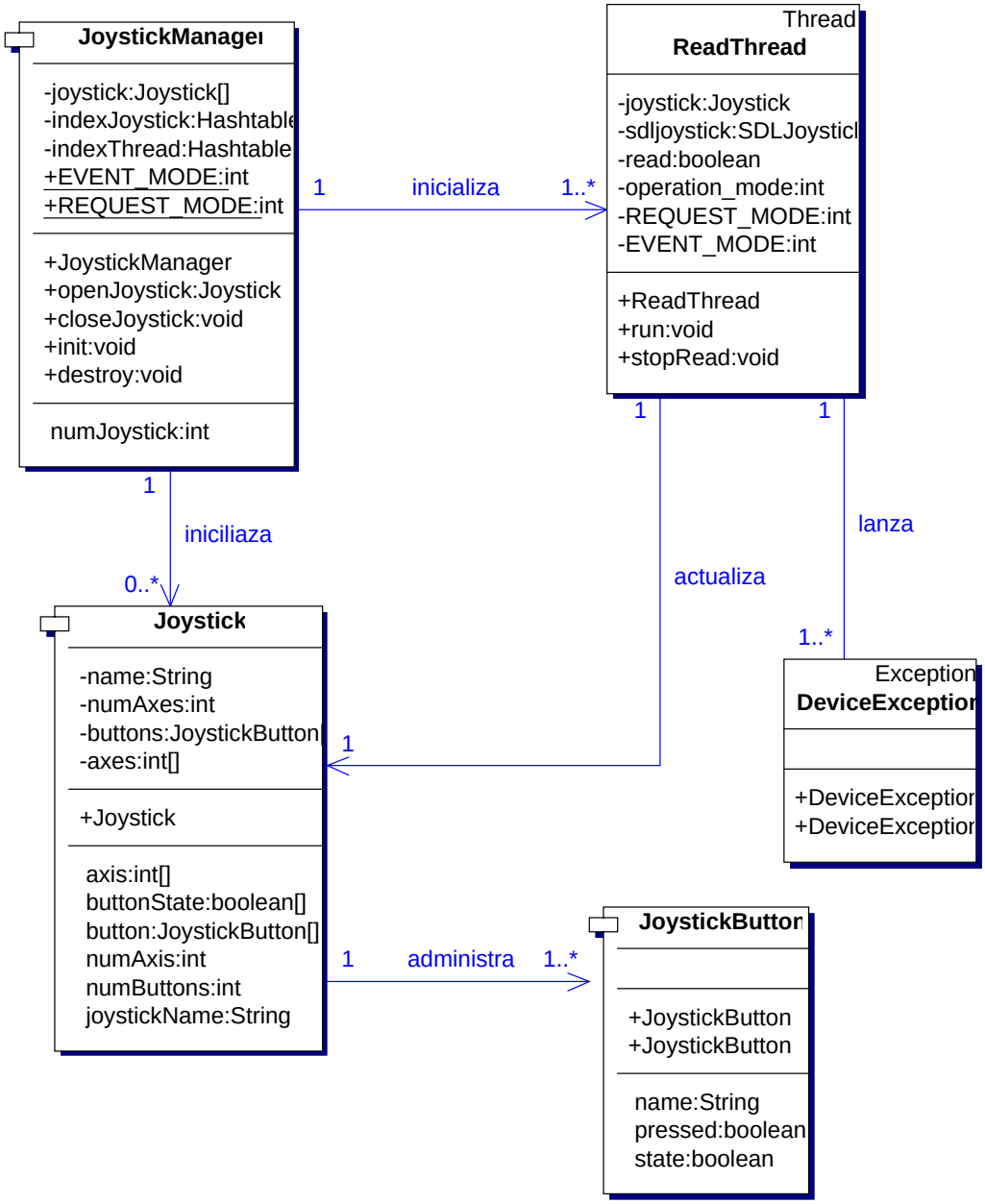


Figura 73 Diagrama de clases del joystick

B3. Guante de Datos

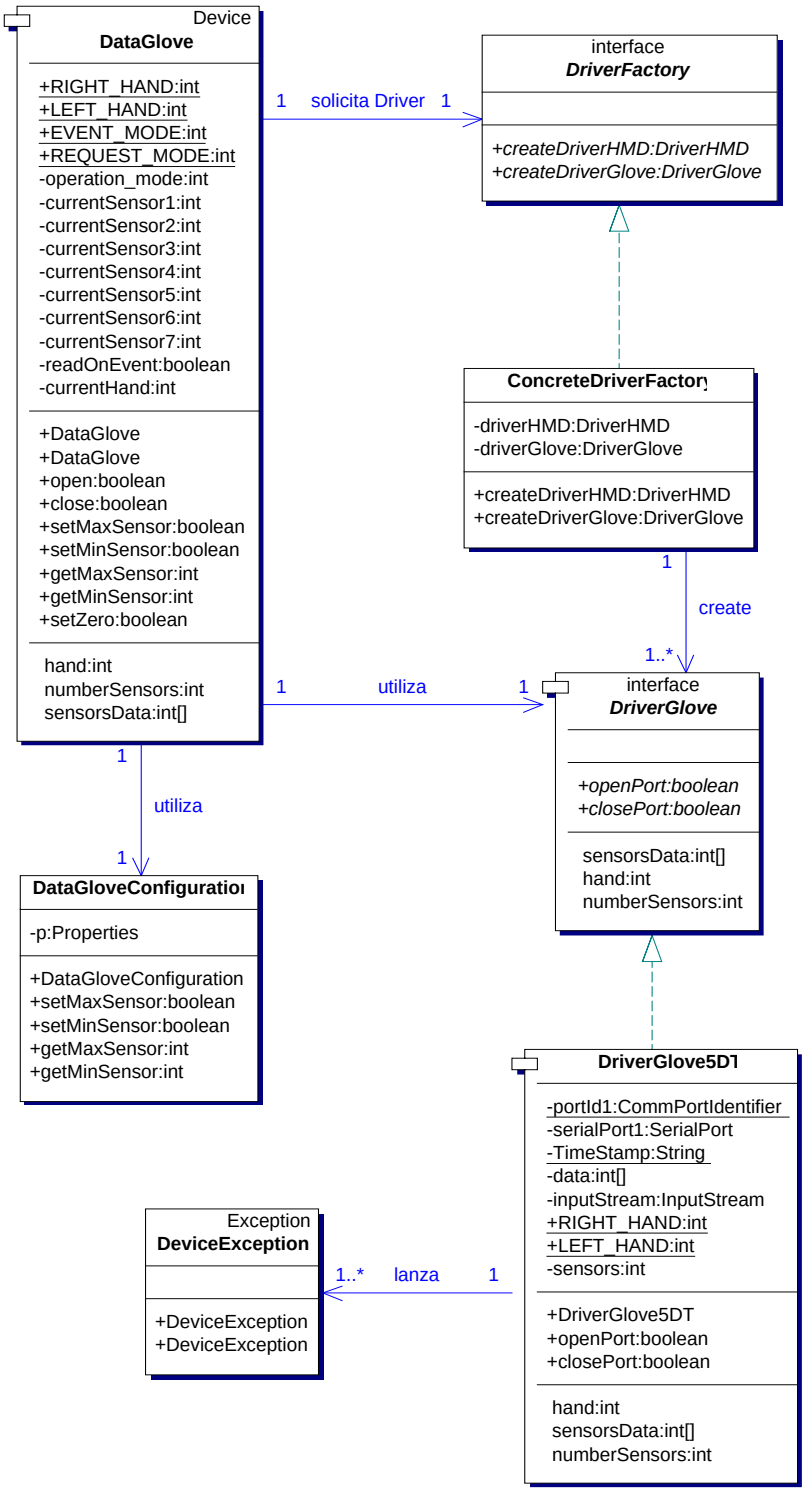


Figura 74 Diagrama de clases a nivel de implementación para guantes de datos

ANEXO C DATOS DE LAS PRUEBAS DEL DRIVER (GUANTE)

Las pruebas del driver desarrollado para el DataGlove 5-Wireless ,consistió básicamente en capturar los paquetes de datos durante un determinado tiempo y obtener el numero de frames capturados en la tabla 9 se muestran los datos para cada una de las pruebas, cada una de ellas difiere entre sí en el tiempo de duración de la prueba, cada dato de la tabla corresponde al número de paquetes obtenidos.

Prueba/Segundos	1	3	5	7	9	11	13	15	17	19
Prueba 1	93	256	454	628	823	1007	1167	1379	1515	1728
Prueba 2	95	270	469	636	812	972	1161	1340	1516	1706
Prueba 3	94	267	461	582	833	1029	1171	1355	1535	1654
Prueba 4	94	285	458	662	793	970	1179	1344	1499	1706
Prueba 5	94	290	461	620	815	942	1160	1369	1530	1695
Prueba 6	93	283	446	639	802	1006	1183	1374	1483	1712
Prueba 7	94	274	459	628	823	997	1193	1348	1538	1704
Prueba 8	93	281	461	630	799	972	1189	1333	1543	1712
Prueba 9	95	262	447	641	807	987	1192	1376	1531	1680
Prueba 10	93	282	464	624	824	948	1121	1367	1529	1719
Prueba 11	95	277	471	619	840	998	1167	1356	1511	1727
Prueba 12	89	252	445	630	764	1010	1160	1321	1502	1667
Prueba 13	94	290	397	642	786	1009	1166	1368	1470	1623
Prueba 14	88	273	448	638	805	963	1177	1313	1505	1682
Prueba 15	93	270	461	634	803	979	1140	1306	1526	1669
Prueba 16	94	286	464	631	726	1010	1074	1376	1456	1683
Prueba 17	92	273	469	630	813	964	1174	1366	1514	1671
Prueba 18	93	281	460	645	830	986	1183	1364	1487	1675
Prueba 19	85	287	451	648	806	996	1194	1357	1489	1655
Prueba 20	85	277	470	638	818	1003	1172	1346	1494	1683

Tabla 9 Paquetes de datos obtenidos durante las pruebas

Como se puede observar en la tabla anterior se realizaron 20 pruebas para cada lapso de tiempo definido, la duración total de la prueba fue alrededor de 4 horas y media.

En la tabla 10 se muestran los valores promediados de cada una de las pruebas realizadas, en donde se puede apreciar en promedio el numero de paquetes obtenidos en cada una de las pruebas.

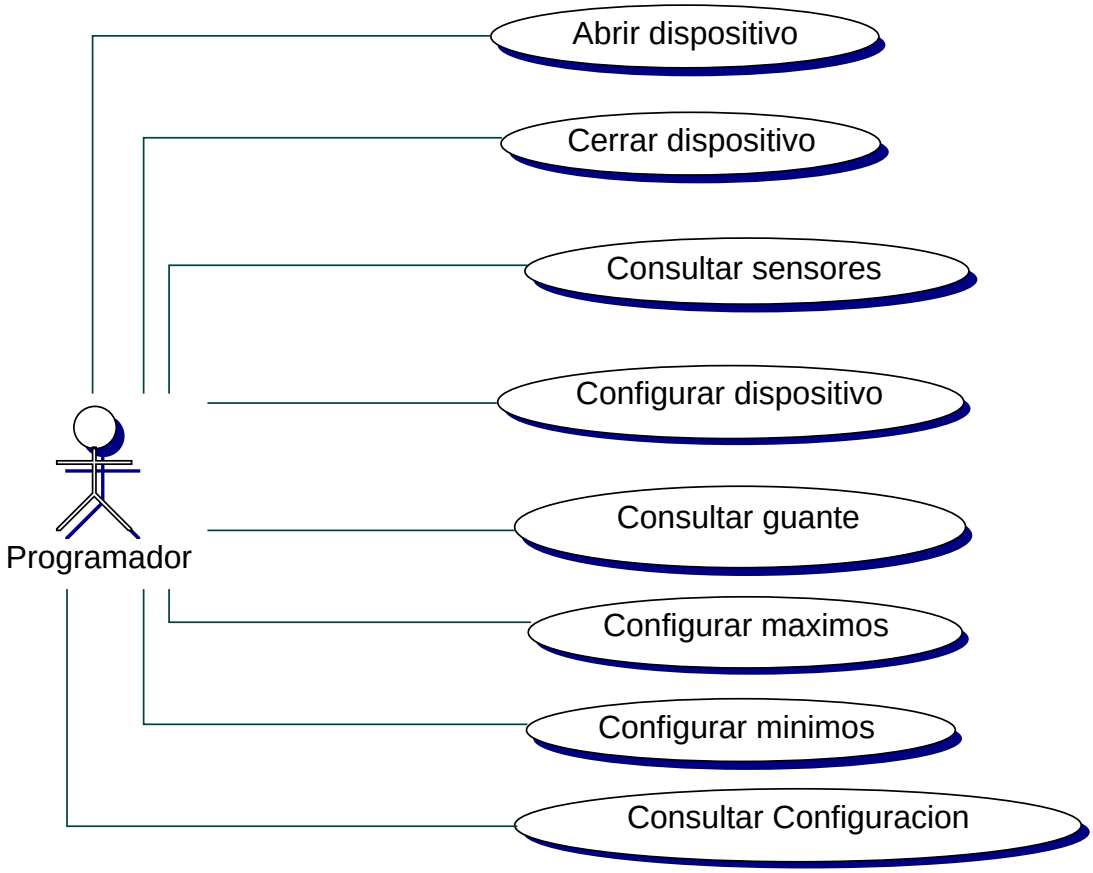
Prueba/Segundos	1	3	5	7	9	11	13	15	17	19
Prueba 1	93	85,3	90,8	89,7	91,4	91,5	89,8	91,9	89,1	90,9
Prueba 2	95	90	93,8	90,9	90,2	88,4	89,3	89,3	89,2	89,8
Prueba 3	94	89	92,2	83,1	92,6	93,5	90,1	90,3	90,3	87,1
Prueba 4	94	95	91,6	94,6	88,1	88,2	90,7	89,6	88,2	89,8
Prueba 5	94	96,7	92,2	88,6	90,6	85,6	89,2	91,3	90	89,2
Prueba 6	93	94,3	89,2	91,3	89,1	91,5	91	91,6	87,2	90,1
Prueba 7	94	91,3	91,8	89,7	91,4	90,6	91,8	89,9	90,5	89,7
Prueba 8	93	93,7	92,2	90	88,8	88,4	91,5	88,9	90,8	90,1
Prueba 9	95	87,3	89,4	91,6	89,7	89,7	91,7	91,7	90,1	88,4
Prueba 10	93	94	92,8	89,1	91,6	86,2	86,2	91,1	89,9	90,5
Prueba 11	95	92,3	94,2	88,4	93,3	90,7	89,8	90,4	88,9	90,9
Prueba 12	89	84	89	90	84,9	91,8	89,2	88,1	88,4	87,7
Prueba 13	94	96,7	79,4	91,7	87,3	91,7	89,7	91,2	86,5	85,4
Prueba 14	88	91	89,6	91,1	89,4	87,5	90,5	87,5	88,5	88,5
Prueba 15	93	90	92,2	90,6	89,2	89	87,7	87,1	89,8	87,8
Prueba 16	94	95,3	92,8	90,1	80,7	91,8	82,6	91,7	85,6	88,6
Prueba 17	92	91	93,8	90	90,3	87,6	90,3	91,1	89,1	87,9
Prueba 18	93	93,7	92	92,1	92,2	89,6	91	90,9	87,5	88,2
Prueba 19	85	95,7	90,2	92,6	89,6	90,5	91,8	90,5	87,6	87,1
Prueba 20	85	92,3	94	91,1	90,9	91,2	90,2	89,7	87,9	88,6
Media	92,3	91,9	91,1	90,3	89,5	89,7	89,7	90,2	88,7	88,8
Desviacion	3,06	3,57	3,21	2,22	2,85	2,09	2,16	1,43	1,39	1,43

Tabla 10 Datos promediados de la prueba

Los resultados anteriores determinaron que el numero promedio de paquetes recibidos es de 90.16 por segundo, sobre un total de **94** frames que envía el guante de datos, por lo tanto en promedio se pierde alrededor de **3.84** frames por segundo lo cual representa el 4.085% del total de frames enviados.

ANEXO D DOCUMENTACION DE CASOS DE USO

D1. (Guantes)



Caso de Uso Abrir Dispositivo

Explicación:

Este caso de uso, maneja la parte de la comunicación, para abrir el puerto y quedar listo para escuchar o enviar los datos hacia el dispositivo.

Pre-condiciones:

La existencia del puerto serial en la maquina en donde se va a probar.

Post-condiciones:

Abre el puerto y queda listo para la comunicación.

Flujo Normal:

a. El usuario envía el puerto que desea abrir
b. El sistema identifica el puerto
c. El sistema verifica si el dispositivo se encuentra conectado.
d. El sistema abre el flujo de datos entre el computador y el dispositivo.
e. El sistema devuelve una notificación de éxito si el puerto fue abierto.

Flujo Alterno

a. Si en el paso 1, el usuario envía un nombre de puerto errado el sistema envía un mensaje de error.

b. Si en el paso 3, encuentra que no esta conectado el dispositivo, el sistema envía un mensaje de error, donde muestra el fracaso abriendo el puerto.

- **Caso de Uso Cerrar Dispositivo**

Explicación:

Cierra el puerto serial para que la comunicación con el dispositivo termine.

Pre-condiciones:

La apertura previa del puerto serial

Post-condiciones:

El éxito de cerrar el puerto y terminar la comunicación entre el computador y el dispositivo.

Flujo Normal:

- El usuario define el puerto a cerrar.
- El sistema identifica el puerto.
- El sistema cierra el puerto deseado.
- El sistema envía un mensaje de éxito al cerrar el puerto.

Flujo Alterno

1. Si el usuario, ingresa mal el puerto el sistema envía un mensaje de error.
2. Si el puerto que se ingresa no existe, el sistema envía un mensaje de notificación de error.

- **Caso de Uso Configurar máximos**

Explicación:

Permite configurar los valores máximos permitidos, de acuerdo a las necesidades del usuario.

Pre-condiciones:

La previa conexión correcta del dispositivo.

Post-condiciones:

Guarda los datos de los valores definidos por el usuario.

Flujo Normal:

1. El usuario pide la solicitud de guardar la configuración.
2. El sistema muestra las posibilidades.
3. El usuario escoge y el sistema los guarda en un archivo.
4. El sistema finaliza el caso de uso.

Flujo Alterno

1. Si los datos no están permitidos el sistema valida y envía un mensaje de error.

- **Caso de Uso Configurar mínimos**

Explicación:

Permite configurar los valores mínimos permitidos, de acuerdo a las necesidades del usuario.

Pre-condiciones:

La previa conexión correcta del dispositivo.

Post-condiciones:

Guarda los datos de los valores definidos por el usuario.

Flujo Normal:

1. El usuario pide la solicitud de guardar la configuración.
2. El sistema muestra las posibilidades.
3. El usuario escoge y el sistema los guarda en un archivo.
4. El sistema finaliza el caso de uso.

Flujo Alterno

1. Si los datos no están permitidos el sistema valida y envía un mensaje de error.

Caso de Uso Consultar configuración

Explicación:

Permite consultar los datos guardados en los casos de uso de configuración de mínimos y máximos

Post-condiciones:

Devuelve los datos guardados en el archivo de configuración.

Flujo Normal:

1. El usuario pide obtener la información de su configuración.
2. El sistema busca el archivo.
3. El sistema toma los datos y los retorna al usuario.

Flujo Alterno

1. Si en el paso 2, el archivo no se encuentra el sistema muestra un mensaje informando la no-existencia y termina.

Caso de Uso Consultar Guante

Explicación:

En este caso de uso, se tiene la información de reconocimiento de que guante esta conectado derecho o izquierdo.

Pre-condiciones:

La previa conexión de un guante y apertura del puerto correctamente.

Post-condiciones:

Devuelve la información de que guante esta conectado.

Flujo Normal:

1. El usuario pide la información de que guante esta conectado.
2. El sistema toma la información del dispositivo.
3. El sistema compara la información y devuelve el dato requerido.
4. El sistema muestra la información.
5. El usuario captura la información y finaliza el caso de uso

Flujo Alterno

1. Si la información para la detección del tipo de guante es errónea, el sistema muestra un mensaje de error.

Caso de Uso Consultar Sensores

Explicación:

Consulta todos los datos de los sensores del guante, y se devuelven al usuario.

Pre-condiciones:

Conexión correcta del dispositivo por el puerto indicado.

Post-condiciones:

Obtención de los datos de los sensores del guante.

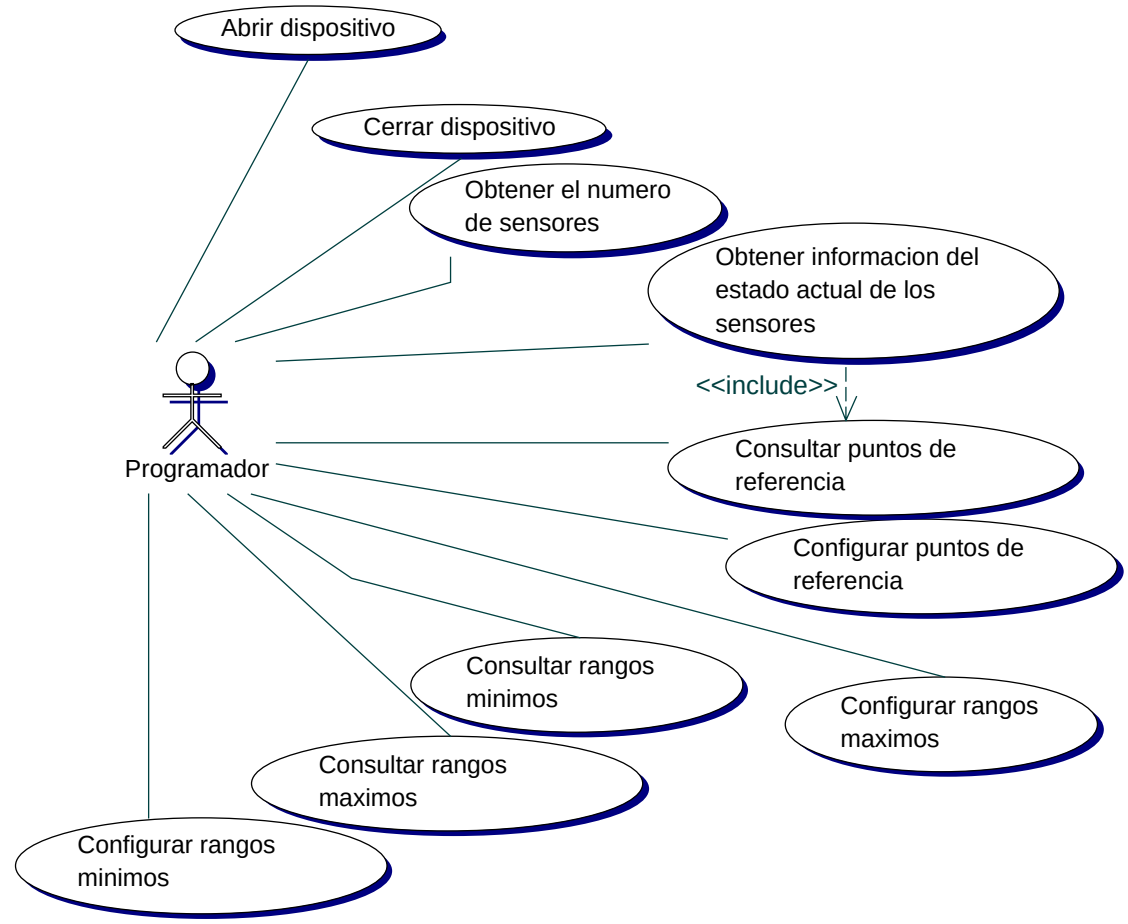
Flujo Normal:

1. El usuario pide la información de los sensores del guante.
2. El sistema pide los datos al dispositivo.
3. El sistema toma los datos y los muestra en un vector.
4. El usuario captura la información del vector y finaliza.

Flujo Alterno

1. Si el dispositivo no envía datos el sistema envía un mensaje de error.

D2. (Cascos)



Caso de Uso Abrir Dispositivo

Explicación:

Permite abrir el dispositivo, en este caso abre el puerto serial para establecer la comunicación con el casco.

Pre-condiciones:

La conexión del dispositivo por el puerto serial.

Post-condiciones:

Deja listo el puerto para enviar o recibir información por el puerto serial.

Flujo Normal:

1. El sistema busca si el dispositivo esta conectado.
2. El sistema establece la comunicación del puerto, de acuerdo al protocolo del dispositivo.
3. El sistema muestra un mensaje de éxito al abrir el puerto.

Flujo Alterno

1. Si el dispositivo no se encuentra conectado, el sistema muestra un error y termina.

Caso de Uso Cerrar Dispositivo

Explicación:

Cierra el puerto serial en donde se encuentra conectado el casco.

Pre-condiciones:

La previa apertura del puerto para la comunicación con el dispositivo.

Post-condiciones:

Cerrar el puerto para que no haya mas flujo de datos.

Flujo Normal:

1. El sistema busca el puerto que se desea cerrar.
2. El sistema envía la información para terminar la comunicación con el dispositivo.
3. El sistema muestra un mensaje de éxito cuando cierra el puerto.

Flujo Alterno

1. Si la información que envía es errónea, el sistema termina.

Caso de Uso Configurar rangos máximos

Explicación:

Permite guardar los datos máximos de los sensores, de acuerdo a la necesidad del programador.

Pre-condiciones:

Previa apertura del puerto y comunicación con el dispositivo.

Post-condiciones:

Guarda el valor deseado por el programador en un archivo plano.

Flujo Normal:

1. El programador pide al sistema que guarde los valores máximos.
2. El sistema despliega las opciones.
3. El programador escoge los valores y da la opción de guardar datos.
4. El sistema almacena los datos en los archivos.
5. El sistema muestra un mensaje de éxito y termina.

Flujo Alterno

1. Si los datos ingresados, son erróneos o no permitidos, el sistema valida y pide el ingreso de los datos.
2. Si el sistema no puede almacenar los datos en el archivo planos, muestra un mensaje de error y termina.

Caso de Uso Configurar rangos mínimos

Explicación:

Permite guardar los valores mínimos de los sensores de acuerdo a la necesidad del programador.

Pre-condiciones:

Previa apertura del puerto y comunicación con el dispositivo.

Post-condiciones:

Guarda el valor deseado por el programador en un archivo plano.

Flujo Normal:

1. El programador pide al sistema que guarde los valores mínimos.
2. El sistema despliega las opciones.
3. El programador escoge los valores y da la opción de guardar datos.
4. El sistema almacena los datos en los archivos.
5. El sistema muestra un mensaje de éxito y termina.

Flujo Alterno

1. Si los datos ingresados, son erróneos o no permitidos, el sistema valida y pide el ingreso de los datos.
2. Si el sistema no puede almacenar los datos en el archivo planos, muestra un mensaje de error y termina.

- **Caso de Uso Consultar rango mínimos**

Explicación:

Permite consultar los datos guardados en el caso de uso de configurar los datos mínimos.

Pre-condiciones:

Previa configuración y conexión con el dispositivo.

Post-condiciones:

Retorna los valores guardados en los archivos de configuración.

Flujo Normal:

1. El programador pide la consulta de los datos.
2. El sistema va y consulta los archivos previamente guardados.
3. El sistema muestra los valores al programador.
4. El sistema termina.

Flujo Alterno

1. Si el sistema no encuentra valores en los archivos, se retorna los valores por omisión.
2. Si el sistema no encuentra los archivos, muestra un error y termina.

- **Caso de Uso Consultar rangos máximos**

Explicación:

Permite consultar los datos guardados en el caso de uso de configurar rangos máximos.

Pre-condiciones:

Previa configuración y conexión con el dispositivo.

Post-condiciones:

Retorna los valores guardados en los archivos de configuración.

Flujo Normal:

1. El programador pide la consulta de los datos.
2. El sistema va y consulta los archivos previamente guardados.
3. El sistema muestra los valores al programador.
4. El sistema termina.

Flujo Alterno

1. Si el sistema no encuentra valores en los archivos, se retorna los valores por omisión.
2. Si el sistema no encuentra los archivos, muestra un error y termina.

Caso de Uso Obtener información actual de los sensores

Explicación:

Consulta los valores de los sensores, dependiendo de los datos configurados o de referencia que tenga.

Pre-condiciones:

Previa apertura del puerto y comunicación con el dispositivo.

Flujo Normal:

1. El programador pide al sistema que le muestre los valores de los sensores.
2. El sistema captura los datos del flujo de datos en la comunicación.
3. El sistema muestra los datos al programador.
4. El sistema termina.

Post-condiciones:

Retorna los datos de los sensores.

Flujo Alterno

1. Si el sistema no puede capturar los datos, muestra un mensaje de error y termina.

Caso de Uso Obtener numero de sensores

Explicación:

Permite reconocer él numero de sensores, que tiene el casco.

Pre-condiciones:

Previa apertura del puerto serial.

Post-condiciones:

Reconocer él numero de sensores que posee el dispositivo.

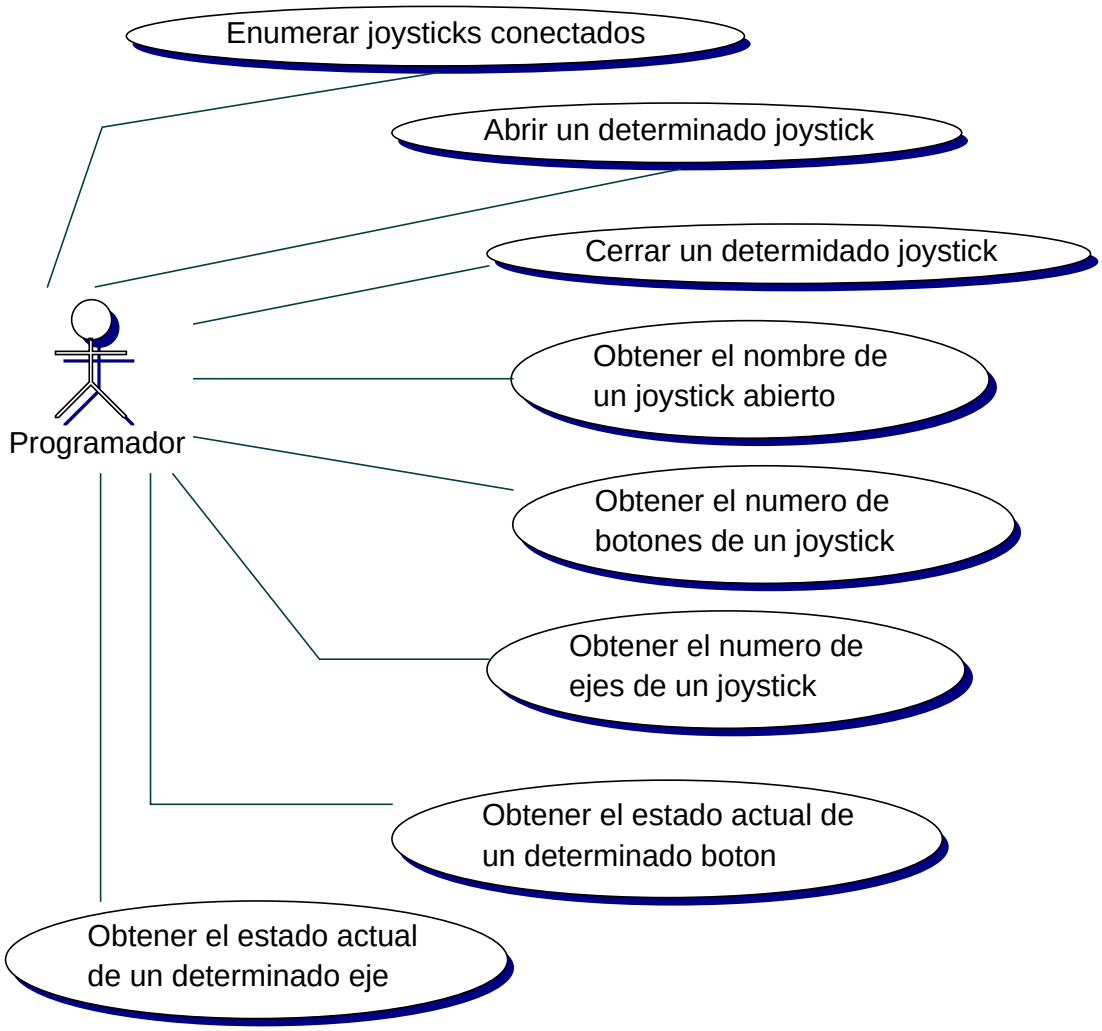
Flujo Normal:

1. El programador pide al sistema que le muestre la información.
2. El sistema pide al dispositivo que devuelva dicha información.
3. El sistema muestra el valor al programador.
4. El sistema termina

Flujo Alterno

1. Si el dispositivo no muestra él numero de sensores, muestra un mensaje de error y termina.

D3. (Joystick, gamepad)



Caso de Uso Abrir un determinado joystick

Explicación:

Este caso de uso permite abrir el puerto para establecer la comunicación con el joystick.

Pre-condiciones:

La conexión de un joystick.

Post-condiciones:

Retorna verdadero o falso si se pudo abrir el puerto.

Flujo Normal:

1. Programador pide el reconocimiento del puerto.
2. El sistema escanea los puertos y determina cual es el que esta abierto con el dispositivo.
3. El sistema termina y muestra un mensaje de éxito.

Flujo Alterno

1. Si el puerto o el dispositivo no esta correctamente conectado, el sistema termina.

- **Caso de Uso Cerrar un determinado joystick**

Explicación:

Este caso de uso cierra la conexión por el puerto, con el joystick.

Pre-condiciones:

La previa conexión del dispositivo.

Post-condiciones:

Mensaje de éxito o fracasado según sea el caso.

Flujo Normal:

1. El sistema escanea los puertos y encuentra el dispositivo que se desea cerrar.
2. El sistema cierra la comunicación con el dispositivo.
3. El sistema muestra un mensaje de éxito y termina.

Flujo Alterno

1. Si el sistema no encuentra el dispositivo, despliega un mensaje de fracaso y termina.

- **Caso de Uso Enumerar joystick conectados**

Explicación:

Este caso de uso, permite enumerar y reconocer los joystick conectados por los puertos USB.

Pre-condiciones:

La previa conexión de algún joystick

Post-condiciones:

Retorna él numero de joysticks conectados.

Flujo Normal:

1. El sistema busca en los puertos los dispositivos conectados.
2. El sistema numera los dispositivos en cada uno de los puertos.
3. El sistema muestra esta numeración.
4. El sistema termina.

Flujo Alterno

1. Si el sistema no encuentra ningún dispositivo conectado, el sistema termina.

- **Caso de Uso Obtener el estado actual de un determinado botón**

Explicación:

Permite reconocer si el botón ha sido oprimido o no.

Pre-condiciones:

La previa apertura de la comunicación con el dispositivo.

Post-condiciones:

Retorna si esta o no presionado un botón o los botones.

Flujo Normal:

1. El programador solicita al sistema la información sobre los botones.
2. El sistema busca el dispositivo.

3. El sistema captura los datos de los botones.
4. El sistema muestra al programador los datos capturados.
5. El sistema termina.

Flujo Alterno

1. Si los datos capturados no son los correctos o no los encuentra, el sistema despliega un mensaje de error y termina

Caso de Uso Obtener el nombre de joystick abierto

Explicación:

Este caso de uso reconoce, que joystick esta conectado.

Pre-condiciones:

La previa apertura de la comunicación con el dispositivo.

Post-condiciones:

Retorna el nombre del dispositivo abierto.

Flujo Normal:

1. El sistema escanea el puerto.
2. El sistema pide la información al dispositivo.
3. El sistema muestra al programador el nombre del dispositivo.
4. El sistema termina.

Flujo Alterno

1. Si el sistema no encuentra el nombre del dispositivo, despliega un mensaje de error y termina.

Caso de Uso Obtener él numero de botones de un joystick

Explicación:

Este caso de uso, retorna él numero de botones del joystick abierto.

Pre-condiciones:

Previa apertura de la comunicación con el dispositivo.

Post-condiciones:

Retorna él numero de botones que posee el dispositivo deseado.

Flujo Normal:

1. El sistema busca el dispositivo.
2. El sistema captura los datos del dispositivo conectado.
3. El sistema muestra al programador él numero de botones con los que cuenta el dispositivo.
4. El sistema termina.

Flujo Alterno

1. Si el sistema no puede tener los datos, despliega un mensaje de error y termina

Caso de Uso Obtener el numero de ejes de un joystick

Explicación:

Este caso de uso permite reconocer cuantos ejes tiene, aunque la mayoría tiene 2, alguno poseen el force feedback.

Pre-condiciones:

Previa apertura de la comunicación con el dispositivo.

Post-condiciones:

Retorna el numero de ejes que maneja el dispositivo en cuestión.

Flujo Normal:

1. El sistema busca el dispositivo.
2. El sistema captura los datos del numero de ejes.
3. El sistema muestra los datos al programador.3. El sistema termina.

Flujo Alterno

1. Si el sistema no puede capturar la información, despliega un mensaje de error y termina.

Caso de Uso Obtener estado actual de un determinado eje

Explicación:

Permite reconocer si la palanca o botones que maneja los ejes han sido oprimidos o soltados.

Pre-condiciones:

Previa apertura de la comunicación con los dispositivos.

Post-condiciones:

Retorna el estado en cuanto a los ejes del dispositivo.

Flujo Normal:

1. El programador solicita al sistema que le retorne el estado de los ejes.
2. El sistema busca el dispositivo.
3. El sistema captura los datos de los ejes.
4. El sistema muestra los datos al programador.
5. El sistema termina.

Flujo Alterno

1. Si los valores capturados son erróneos, el sistema muestra error y termina.

ANEXO E. DIAGRAMAS DE PATRONES DE DISEÑO

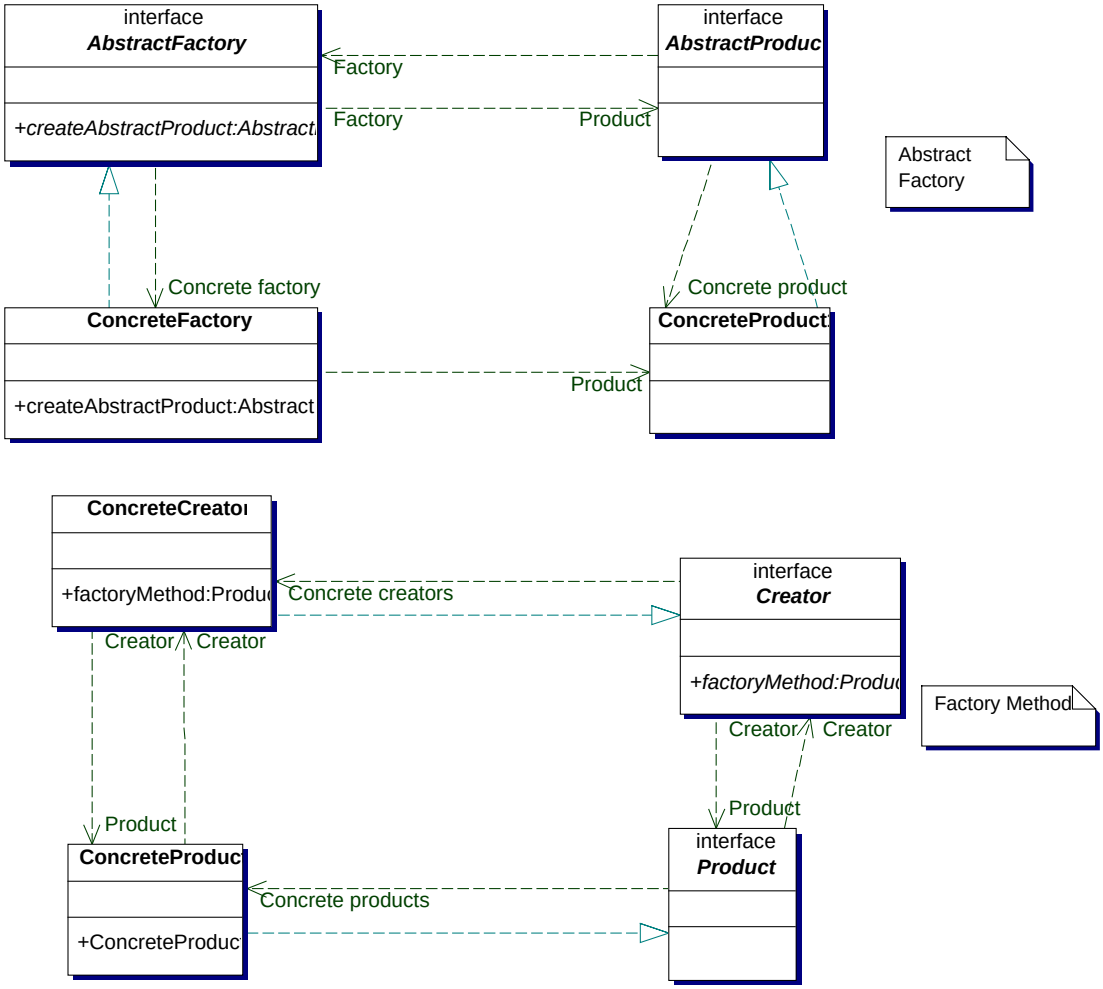


Figura 75 Patrones de diseño

ANEXO F. DOCUMENTACIÓN PUENTE NATIVO

Para establecer la comunicación a través del puerto serial se construyó una solución nativa en C++, esto con el único objetivo de configurar la velocidad del puerto a 14400 bps, ya que Java Communications API no la soporta. Para poder emplear código nativo desde una aplicación Java debe utilizarse JNI (Java Native Interface), la cual define los siguientes pasos:

1. Se debe crear una clase Java en la que se deben definir los métodos que se implementaran en código nativo, dichos métodos deben estar precedidos por la palabre reservada ***native.*** La definición de los métodos del puente nativo se muestran a continuación:

```
native int open_port(String port);
native int[] readData(int fd,int size);
native int writeData(int fd,int[] data,int length);
native int close_port(int fd);
native void setBaudRate(int fd,int baud);
```

2. Se debe compilar la clase que contiene los métodos nativos.
3. Se debe generar el archivo de *header* mediante la herramienta **javah** con la opción **–jni**, en el puente nativo dicho archivo de encabezado se denomina **HMDSerialComm.h,** las firmas de los métodos de este archivo se muestran a continuación:

```
JNIEXPORT jint JNICALL Java_HMDSerialComm_open_1port(JNIEnv *, jobject, jstring);

JNIEXPORT jintArray JNICALL Java_HMDSerialComm_readData(JNIEnv *, jobject, jint,
jint);

JNIEXPORT jint JNICALL Java_HMDSerialComm_writeData(JNIEnv *, jobject, jint,
jintArray, jint);

JNIEXPORT jint JNICALL Java_HMDSerialComm_close_1port(JNIEnv *, jobject, jint);

JNIEXPORT void JNICALL Java_HMDSerialComm_setBaudRate(JNIEnv *, jobject, jint,
jint);
```

4. Una vez generado el archivo de encabezado se debe construir el archivo de implementación el cual debe tener extensión **.c** o **.cpp,** en donde se debe escribir el código correspondiente a cada uno de los métodos, en el puente nativo la implementación de estos métodos se encarga de realizar el mapeo

de los datos definidos en la interfaz **JNI.h** a los datos propios del lenguaje **C++**, posteriormente se llama a los métodos que contienen el acceso a el puerto serial, los cuales se encuentran en el archivo **DriverHMD.cpp**

5. Al finalizar la implementación nativa de los métodos se debe generar un archivo .DLL mediante la siguiente instrucción en la línea de comandos:

```
cl -Ic:\java\include -Ic:\java\include\win32
–LD HMDSerialComm.cpp -libserialnative.dll
```

6. Ahora debe crearse un bloque estático en la clase Java que define los métodos nativos , en este bloque debe cargarse el archivo DLL generado en el paso anterior, para asegurarse que el archivo es encontrado debe ubicarse en la misma carpeta donde se encuentra los archivos **.class** ó se puede copiar el archivo en la carpeta system32. El bloque estático debe poseer el llamado a la función System.loadLibrary(), como se muestra a continuación:

```
static
{
       System.loadLibrary("libserialnative");
}
```

Para emplear el puente nativo se construyó la clase **DriverHMDVFX3D** en la que se realiza el manejo del FileDescriptor, en esta clase también se encuentra el protocolo de comunicación y el mapeo de los datos de las tramas de información que se reciben del casco VFX3D a los valores lógicos que necesita la librería.