Desarrollo de componentes InTml (Interaction Techniques Markup Language) sobre Java con Java3D

Nicolás Mendoza Ruíz

Departamento de ingeniería de sistemas - Universidad de los Andes

21/06/2011

TABLA DE CONTENIDO

0 Resumen 4

1 Introduccion 4

2 Estándar de nombramiento 5

2.1 Un puerto de entrada que requiere puerto de salida 5

2.2 Banderas de activación 6

2.3 Cantidad de datos deseada en un puerto 6

2.4 Combinación de prefijos 6

3 Especificación de filtros 7

Tracker 7

TrackerSelector 8

OffSetter 8

PosToPosOrient 9

RayCaster 9

FeedbackToggle 10

ObjectSelector 11

Scene 11

SimpleObject 12

Ray 13

4 Implementacion 14

4.1 Esquema general de la implementación de cada filtro 14

4.1.1 Comportamiento 14

4.1.1.1 Estructura de la clase 15

4.1.2 Integración 16

3.1.2.1 Estructura de la clase 17

4.1.3 Dependencia circular 21

4.2 Descripción especifica de la implementación de cada filtro 22
Tracker 22
TrackerSelector 23
OffSetter 24
PosToPosOrient 24
RayCaster 25
FeedbackToggle 25
ObjectSelector 26
Scene 26
SimpleObject 26
Ray 28

0 RESUMEN

InTml es un lenguaje visual que permite crear aplicaciones de realidad virtual o realidad mixta a personas no familiarizadas con desarrollo de software. Para facilitar la creación de aplicaciones sobre InTml es necesaria la existencia de librerías de componentes (en adelante filtros) prediseñados e implementados de tal forma que una persona pueda crear una aplicación utilizando los componentes ya existentes. La motivación del trabajo desarrollado fue precisamente ampliar la cantidad de componentes existentes para las librerías estándar de InTml mediante la implementación de dos técnicas de interacción: "Ray Casting" y "Two Handed Pointing".

1 INTRODUCCION

Para la creación de una aplicación de realidad virtual o realidad mixta se necesita de conocimientos en computación gráfica y desarrollo de software que permitan la implementación de la idea que se quiera plasmar. La profundidad en conocimientos requeridos varía dependiendo de la complejidad y el tamaño de la aplicación que se quiera hacer. Además de los conocimientos requeridos, se necesita de tiempo de implementación que puede llegar a extenderse indeseablemente como en cualquier proyecto de desarrollo de software. Mucho de ese tiempo empleado en implementación se gasta en problemas que son comunes a aplicaciones de este tipo sin embargo las soluciones suelen estar acopladas fuertemente a la aplicación o a la tecnología que se esté usando y por lo tanto su reutilización se dificulta. Utilizando InTml se propone una solución en donde personas ajenas al desarrollo de software puedan crear sus propias aplicaciones de realidad virtual si se les da una introducción al lenguaje. Para que se puedan crear dichas aplicaciones sin la necesidad de habilidades en computación gráfica y desarrollo de software se necesita tener una librería de componentes estándar que solucionen problemas comunes a aplicaciones de este estilo. De esta forma el diseñador de la aplicación solo necesita conectar los filtros que vaya a usar en la aplicación.

Los filtros que se desarrollen deben ser hechos de tal forma que se logre la mayor independencia posible con respecto al runtime de InTml. De esta forma un desarrollador puede desarrollar las funcinalidades de un filtro sin tener la necesidad de conocer la estructura del runtime, posteriormente se puede hacer la integración con InTml. Lo que se debe tener claro a la hora de desarrollar un filtro es el modelo de ejecución de InTml. Si no se conoce como se propagan los datos entre filtros, el desarrollo de cualquier filtro puede ser malo.

El resultado del proyecto fue la implementación de dos técnicas de interacción: Ray casting y Two handed pointing. Para el movimiento de la cámara se desea que este sea controlado por el movimiento de la cabeza del usuario, mientras que el movimiento del rayo se desea que sea controlado por las manos del usuario (una sola mano para el caso de Ray casting). Las dos técnicas fueron probadas con dos dispositivos: Microsoft Kinect y una cámara OptiTrack FLEX:V100R2 con un TrackClip adherido a una gorra. En primera instancia se probó la funcionalidad de las dos técnicas de interacción utilizando únicamente Microsoft Kinect para rastrear los movimientos de la cabeza y de las manos. Lo que se observó es que el movimiento de las manos interfiere con el movimiento de la cabeza cuando se utiliza Kinect solamente y por la tanto la interacción no es buena. Utilizando la combinación de Kinect y la cámara OptiTrack, la interferencia se eliminó y los dos movimientos se independizaron. Sin embargo, se observo ruido en el movimiento de la cámara y del rayo. Cuando la cabeza del usuario permanecía estática, se observó un movimiento ligero de la cámara que no se desea en la experiencia interactiva. Así mismo, por las imprecisiones del rastreo de las manos con Kinect, para lograr la interacción deseada los movimientos de las manos se deben exagerar un poco.

Estructura del documento

El contenido del documento se divide en tres partes principales: estándar de nombramiento, especificación de filtros e implementación. Para entender de una mejor forma el documento, es

mejor leerlo en el orden que se presentan las tres secciones, ya que el estándar de nombramiento provee una base para entender mejor el comportamiento de cada filtro y la especificación de cada filtro sirve como referencia para entender la implementación de cada filtro. En la sección de implementación se muestra la forma general como se desarrollaron todos los filtros y después se presentan los detalles de la implementación de cada filtro

Agradecimientos

Le agradezco mucho al profesor Pablo Figueroa por la guía y el apoyo que me brindó durante todo el semestre, respondiendo a todas las inquietudes que surgieron en el camino y brindando un apoyo constante durante todo el proceso de desarrollo. También le agradezco mucho a Esteban Correa quien colaboró constantemente y aportó mucho a la implementación final, y me acompañó durante muchos ratos durante el semestre.

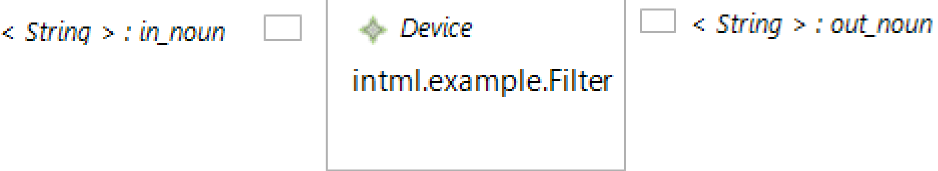
2 ESTÁNDAR DE NOMBRAMIENTO

Durante el desarrollo de filtros, se encontraron varias dificultades al momento de entender el comportamiento de cada filtro. Dado que la librería de filtros se encontraba diseñada previamente, pero no documentada, se realizó la especificación de una buena cantidad de filtros, sin embargo la funcionalidad de algunos filtros no era explicita. Es claro que entre más intuitivos sean los filtros, más fácil será diseñar una aplicación en InTml. Para realizar una analogía, cuando un programador se enfrenta a un API que nunca haya usado en su vida, para él será más rápido desarrollar una aplicación si los métodos o funciones (y sus parámetros) que utilice en dicho API tienen un nombre claro y acorde con la función que desempeñan. Por tal motivo se pensó en un estándar de nombramiento aplicable a todos los filtros con el fin de que la funcionalidad de cada filtro y la cantidad de datos que arroja sean más fáciles de entender a simple vista.

A continuación se enumeran las consideraciones que se deben tener en cuenta en el momento del nombramiento de cada filtro.

2.1 Un puerto de entrada que requiere puerto de salida

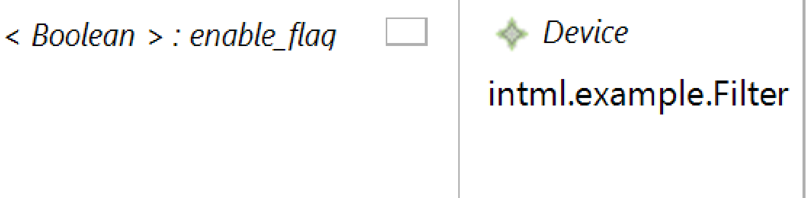
En ocasiones, se pueden presentar situaciones en donde se necesita saber un valor (ya sea un tipo básico o un filtro que se utiliza como valor de entrada o salida) que está siendo utilizado en el filtro. Si dicho valor es una réplica o una modificación (ej. La selección de un valor entre varios valores recibidos por un puerto de entrada en un determinado frame) de un valor o un conjunto de valores que se reciben por un puerto de entrada, se utiliza el siguiente estándar:



En donde “noun” se reemplaza por un sustantivo que caracteriza al puerto (ej. position). Cabe anotar que la palabra que va a ser reemplazada por “noun” debe ser escrita en minúsculas teniendo en cuenta que si es una combinación de palabras, en vez de utilizar un espacio para separar las palabras, la primera letra de la palabra que siga debe ser mayúscula así: *palabraPalabraPalabra…* y los prefijos utilizados (*in_, out_*) van en minúscula. Adicionalmente el tipo de datos de ambos puertos debe ser el mismo. En la ilustración se utiliza el tipo String como ejemplo.

2.2 Banderas de activación

Cuando se requiera que un puerto active algún comportamiento dentro del filtro, el estándar a utilizar es el siguiente:



Para este estándar se utiliza el prefijo *enable_* seguido de la palabra que indica el comportamiento que se quiere activar. En este caso la palabra que está siendo utilizada es "flag". Al igual que en la consideración anterior, la palabra por la cual se va a reemplazar "flag" debe estar en minúsculas, con la excepción de la primera letra de las palabras intermedias, y el prefijo debe estar en minúsculas.

2.3 Cantidad de datos deseada en un puerto

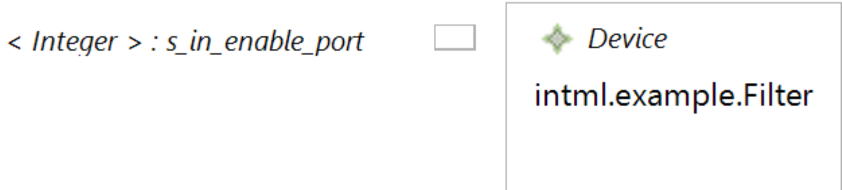
A pesar de que en el modelo de Data Flow de InTml siempre puede llegar un número indeterminado de valores a cada puerto, hay filtros que utilizan solo un valor en el puerto para llevar a cabo su función. Por ejemplo en el caso hipotético de un filtro que sirva para sumar dos números, debería haber dos puertos de entrada, cada uno representando uno de los números a sumar. Si en uno de los puertos entran 100 números diferentes y en el otro puerto entran 300 números diferentes, el comportamiento esperado del filtro no es que sume los 100 números de un puerto con los primeros 100 números del otro puerto, sino que escoja un número entre los 100 (también podría hacer un promedio u otra función dependiendo de lo que se requiera) que entraron por un puerto y otro número entre los 300 que entraron por otro puerto y sume los números escogidos. Para distinguir el tipo de comportamiento deseado en el puerto se utiliza el siguiente estándar:

< Integer > : s_port1
< Integer > : m_port2
Device
intml.example.Filter
< Integer > : s_port3
< Integer > : m_port4

Como se puede ver en la figura, se utilizan dos tipos de prefijos: *s_, m_*. Si se quiere denotar que el filtro utiliza un solo dato en el puerto de entrada, se utiliza el prefijo *s_* (de "single"). De forma contraria, si se utilizan todos los datos que entran por el puerto, se utiliza el prefijo *m_* (de "multiple"). De igual forma, si en los puertos de salida se espera que el filtro arroje un solo dato se utiliza *s_*, y si se espera arrojar múltiples datos se utiliza *m_*.

2.4 Combinación de prefijos

En ocasiones se desea combinar el comportamiento descrito en las consideraciones de arriba.
Para ese caso se utiliza el siguiente orden:



De esta forma el orden que se debe seguir es:

I. Prefijo de cantidad de datos esperada
II. Prefijo de relación entrada/salida
III. Prefijo de activación de bandera

3 ESPECIFICACIÓN DE FILTROS

A continuación se muestra el listado de todos los filtros implementados para las técnicas “Ray casting” y “Two handed pointing”. Para cada filtro se especifican entradas, salidas, descripción de su funcionalidad y se muestra el diagrama del filtro.

Tracker

Filtro que describe un dispositivo genérico de tipo tracker con varios elementos de entrada. El dispositivo arroja valores tridimensionales tanto de orientación como de posición.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_connInfo	String	Cadena que contiene la información para conectarse al tracker deseado por medio de VRPN.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
s_deviceName	String	Cadena que representa el identificador del tracker asociado a este filtro.
m_positionValues	IdPos3D	Flujo de datos de posición que arroja el dispositivo.
m_rotationValues	IdOrientation3D	Flujo de datos de orientación que arroja el dispositivo.
s_numPositions	Integer	Número de posiciones que entrega el tracker.
s_numRotations	Integer	Número de orientaciones que entrega el tracker.

Diagrama del filtro

< String > : s_connInfo
Device
intml.devices.library.Tracker
< String > : s_deviceName
< IdPos3D > : m_positionValues
< IdOrientation3D > : m_rotationVal...
< Integer > : s_numPositions
< Integer > : s_numRotations

TrackerSelector

Su función consiste en seleccionar los datos de un elemento de entrada (e.g posición y orientación de la mano izquierda) de un Tracker. Los datos de entrada son los flujos de datos de los trackers y números enteros para identificar los datos que se quieren seleccionar. Los datos de salida son datos de posición y orientación sin identificadores.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_desiredPosition	Integer	Contiene el identificador de la posición que se desea seleccionar.
s_desiredOrientation	Integer	Contiene el identificador de la orientación que se desea seleccionar.
m_in_position	IdPos3D	Es el flujo de datos de posición proveniente del tracker.
m_in_orientation	IdOrientation3D	Es el flujo de datos de orientación proveniente del tracker.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
s_out_position	Pos3D	Contiene los valores de posición seleccionados a partir del flujo de datos del tracker.
s_out_orientation	Orientation3D	Contiene los valores de orientación seleccionados a partir del flujo de datos del tracker.

Diagrama

< Integer > : s desiredPosition
< Integer > : s desiredOrientation
< IdPos3D > : m in position
< IdOrientation3D > : m in orienta...
Behavior
intml.devices.library.TrackerSelector
< Pos3D > : s out position
< Orientation3D > : s out orientation

OffSetter

Transforma una posición y una orientación utilizando varios valores de entrada. La posición es transformada aplicándole una traslación y una escala, y la orientación es transformada aplicándole una rotación. Los valores con los cuales se transforman los datos son mantenidos hasta que se reciban nuevos valores.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_in_position	Pos3D	Contiene la coordenada que se va a transformar.
s_in_orientation	Orientation3D	Contiene la orientacion que se va a transformar.
s_positionOffset	Pos3D	Es la translación que se va a aplicar en la transformación de la posición.
s_orientationOffset	Orientation3D	Es la rotación que se va a aplicar en la transformación de la orientación.
s_scaleOffset	Integer	Es la escala que se va a aplicar en la posición a transformar.

Puertos de salida

Nombre	Tipo de dato	Descripción
s_out_position	Pos3D	El resultado de la transformación de posición se arroja por este puerto.
s_out_orientation	Orientation3D	El resultado de la transformación de orientación se arroja por este puerto.

Diagrama

< Pos3D > : s in position
< Orientation3D > : s in orientation
< Pos3D > : s positionOffset
< Orientation3D > : s orientationO...
< Float > : s scaleOffset
Behavior
intml.selection.library.OffSetter
< Pos3D > : s out position
< Orientation3D > : s out orientation

PosToPosOrient

Toma dos posiciones y arroja una posición y una orientación basándose en las dos posiciones. La posición resultante es la misma posición 1 de entrada y la orientación es el vector que resulta de trazar una flecha desde la posición 1 a la posición 2.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_pos1	Pos3D	La primera posición para calcular la posición y orientación final.
s_pos2	Pos3D	La segunda posición para calcular la posición y orientación final.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
s_position	Pos3D	La posición resultante. Es igual a la posición 1.
s_orientation	Orientation3D	El resultado del cálculo de la orientación.

Diagrama

< Pos3D > : s pos1
< Pos3D > : s pos2
Behavior
intml.selection.library.PosToPosOrient
< Pos3D > : s position
< Orientation3D > : s orientation

RayCaster

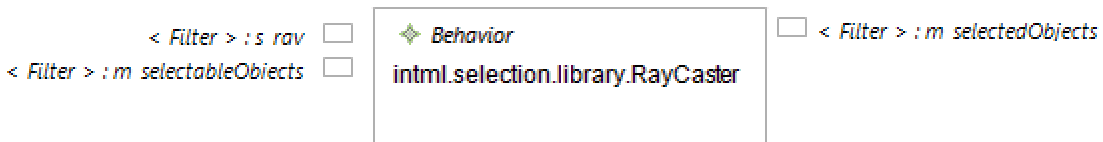
Selecciona objetos por medio de un rayo. Recibe como entrada el rayo y los objetos que pueden ser seleccionados. Si algún objeto es atravesado por el rayo, este objeto sale por el puerto de salida del filtro.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_ray	Filter	El rayo por medio del cual se van a seleccionar los objetos
m_selectableObjects	Filter	Los objetos que pueden ser seleccionados por el rayo.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
m_selectedObjects	Filter	Los objetos que fueron seleccionados por el rayo.

Diagrama



FeedbackToggle

Cambia la apariencia de los objetos recibidos según el estilo seleccionado para dar retroalimentación visual de alguna acción. El estilo puede ser “bounding box o “material” (bbox, material). Una vez que se reciben nuevos objetos, los objetos anteriores cambian su apariencia a su estado normal.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
m_objects	Filter	Puerto de entrada para los objetos cuya apariencia va a ser cambiada.
s_enable	Boolean	Bandera para activar o no el cambio de apariencia de los objetos.
s_style	String	El estilo que se va a aplicar al momento de cambiar la apariencia de los objetos. Puede ser: bbox o material.

Diagrama

< Filter > : m objects
< Boolean > : s enable
< String > : s style
Behavior
intml.selection.library.FeedbackToggle

ObjectSelector

Selecciona objetos entrantes por un flujo de datos de un puerto dado un identificador o un grupo de identificadores de los objetos. Si es un grupo de identificadores, en la cadena de entrada, estos deben estar separados por espacios. El resultado de la selección se arroja por dos puertos, uno para los objetos seleccionados y otro para los no seleccionados.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
m_in_objects	Filter	Flujo de objetos a seleccionar.
m_objectIds	String	Cadena por donde se van a recibir los identificadores de los objetos. Si son varios identificadores, deben estar separados por espacios.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
m_out_selectedObjects	Filter	Flujo de objetos seleccionados por el filtro.
m_out_nonSelectedObjects	Filter	Flujo de objetos no seleccionados.

Diagrama

< Filter > : m in objects
< String > : m objectIds
Behavior
intml.objects.library.ObjectSelector
< Filter > : m out selectedObjects
< Filter > : m out nonSelectedObjects

Scene

Este filtro carga los objetos que se van a visualizar en la aplicación y los entrega por un puerto. Todos los objetos son visibles inicialmente. Los objetos son cargados de un archivo especificado por una ruta. Actualmente solo se soporta VRML 2.0 y para que un nodo del archivo sea considerado un objeto, se debe especificar un nombre para el objeto en el campo DEF del nodo. El nombre debe estar antecedido por el prefijo “Object_”.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_filePath	String	La ruta en donde se encuentra el archivo que describe la escena de la aplicación. Actualmente solo se soporta VRML 2.0.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
m_objectStream	Filter	Flujo de objetos cargados.

Diagrama

< String > : s_filePath   Object   intml.objects.library.Scene   < Filter > : m_objectStream

SimpleObject

Filtro que representa un objeto de la escena de la aplicación. Este tipo de filtro es el que se arroja por el puerto “m_objectStream” del filtro Scene. El filtro contiene varios puertos de entrada que sirven para modificar propiedades comunes a objetos en una escena, como lo son su posición, su oientación, el escalamiento que se le deba aplicar, si el objeto es seleccionable, etc. El objeto también puede ser cargado independientemente del filtro escena.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_fileName	String	Ruta del archivo que contiene la información geométrica del objeto.
s_in_position	Pos3D	Puerto de entrada para especificar la posición del objeto.
s_in_orientation	Orientation3D	Puerto de entrada para especificar la orientación del objeto.
s_in_scale	Float	Número que representa el escalamiento que se le debería aplicar al objeto.
s_enable_visible	Boolean	Bandera para especificar si el objeto debe ser visible o no.
s_enable_showBBox	Boolean	Bandera para activar y mostrar el bounding box del objeto.
s_material	Filter	Puerto para especificar el material que deba tener este objeto.
s_enable_useMaterial	Boolean	Bandera que indica si se debe utilizar el material que haya entrado por el puerto “s_material”.
s_in_parent	Filter	Puerto utilizado para especificar un objeto padre para este objeto.
s_enable_removeParent	Boolean	Bandera para remover el padre del objeto si se encuentra activada.
s_enable_selectable	Boolean	Bandera para indicar si el objeto puede ser seleccionado o no.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
s_out_position	Pos3D	Arroja la posición actual del objeto.
s_out_orientation	Orientation3D	Arroja la orientación actual del objeto.
s_out_scale	Float	Entrega el escalamiento actual del objeto.
s_out_parent	Filter	Es el objeto padre de este objeto, si existe.
s_id	String	El identificador del objeto.

Diagrama

< String > : s fileName
< Pos3D > : s in position
< Orientation3D > : s in orientation
< Float > : s in scale
< Boolean > : s enable visible
< Boolean > : s enable showBBox
< Filter > : s material
< Boolean > : s enable useMaterial
< Filter > : s in parent
< Boolean > : s enable removePare...

Object
intml.objects.library.SimpleObject

< Pos3D > : s out position
< Orientation3D > : s out orientation
< Float > : s out scale
< Filter > : s out parent
< String > : s id

Ray

Es el objeto que representa un rayo. El objeto rayo es utilizado en las técnicas de interacción “Ray casting” y “Two handed pointing”. Su propiedades son muy parecidas a las de SimpleObject y en caso de no especificar una ruta para el objeto que se deba visualizar, por defecto se mostrará un cono que representará el rayo y la dirección hacía la cual se está apuntando.

Puertos

Puertos de entrada		
Nombre	**Tipo de dato**	**Descripción**
s_fileName	String	Ruta del archivo que contiene la información geométrica del objeto.
s_in_position	Pos3D	Puerto de entrada para especificar la posición del objeto.
s_in_orientation	Orientation3D	Puerto de entrada para especificar la orientación del objeto.
s_in_scale	Float	Número que representa el escalamiento que se le debería aplicar al objeto.
s_enable_visible	Boolean	Bandera para especificar si el objeto debe ser visible o no.
s_enable_showBBox	Boolean	Bandera para activar y mostrar el bounding box del objeto.
s_material	Filter	Puerto para especificar el material que deba tener este objeto.
s_enable_useMaterial	Boolean	Bandera que indica si se debe utilizar el material que haya entrado por el puerto “s_material”.
s_in_parent	Filter	Puerto utilizado para especificar un objeto padre para este objeto.
s_enable_removeParent	Boolean	Bandera para remover el padre del objeto si se encuentra activada.
s_enable_selectable	Boolean	Bandera para indicar si el objeto puede ser seleccionado o no.
s_enable_useLenght	Boolean	Bandera que indica si el rayo debe tener una longitud finita al momento de seleccionar objetos con este.
s_in_length	Float	Longitud del rayo.
s_in_radius	Float	Radio del rayo.

Puertos de salida		
Nombre	**Tipo de dato**	**Descripción**
s_out_position	Pos3D	Arroja la posición actual del objeto.
s_out_orientation	Orientation3D	Arroja la orientación actual del objeto.
s_out_scale	Float	Entrega el escalamiento actual del objeto.
s_out_parent	Filter	Es el objeto padre de este objeto, si existe.

s_id	String	El identificador del objeto.
s_out_length	Float	Longitud actual del rayo.
s_out_radius	Float	Radio actual del rayo.

Diagrama

< String > : s fileName
< Pos3D > : s in position
< Orientation3D > : s in orientation
< Float > : s in scale
< Boolean > : s enable visible
< Boolean > : s enable showBBox
< Filter > : s material
< Boolean > : s enable useMaterial
< Filter > : s in parent
< Boolean > : s enable removePare...
< Boolean > : s enable selectable
< Boolean > : s enable useLength
< Float > : s in length
< Float > : s in radius

Object
intml.objects.library.Ray

< Pos3D > : s out position
< Orientation3D > : s out orientation
< Float > : s out scale
< Filter > : s out parent
< String > : s id
< Float > : s out length
< Float > : s out radius

4 IMPLEMENTACION

A continuación se describe de forma general la forma en cómo se implementó cada filtro seguido de detalle específicos de implementación de cada filtro.

4.1 Esquema general de la implementación de cada filtro

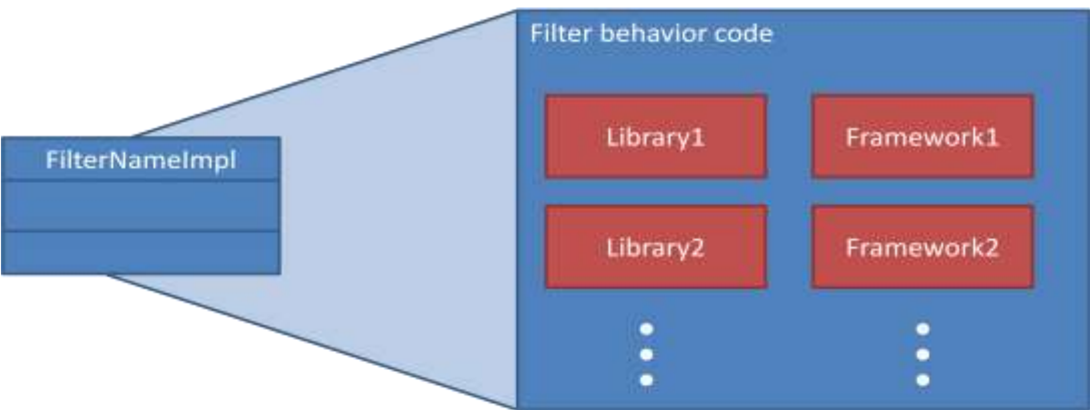
En esta sección se describe la forma general en la que se desarrollaron todos los filtros expuestos en este documento. La implementación de cada filtro se dividió en dos partes: parte de comportamiento y parte de integración. Cada una de las partes se ve reflejada en la implementación de dos clases diferentes a las cuales llamaremos clase de comportamiento y clase de integración.

4.1.1 Comportamiento

La parte de comportamiento del filtro es la responsable de ejecutar las tareas para las cuales fue diseñado el filtro. Para que se pueda desarrollar una implementación que sea independiente del ambiente de ejecución de InTml, esta parte no debe estar acoplada con el framework del ambiente de ejecución de InTml, sin embargo, más adelante se mostrará que es necesaria una dependencia circular con InTml en un punto. Esta dependencia circular debe ser la única dependencia que se debería tener en la parte de comportamiento. La implementación de los filtros debe ser lo más independiente posible y por lo tanto solo debe utilizar frameworks y librerías externas en la parte de comportamiento.

La parte de comportamiento se compone de una clase que va a encapsular todo lo que debería ejecutar el filtro. El nombre de dicha clase es NombreFiltroImpl, en donde “NombreFiltro” representa el nombre del filtro asociado a dicha clase e “Impl” (el sufijo proviene de la palabra implementation) es un sufijo del nombre de la clase para identificar que se trata de la clase de comportamiento. Por lo tanto, si se tiene un filtro de nombre OffSetter, su respectiva clase de

comportamiento es OffSetterImpl. Como se muestra en la figura de abajo, la clase de comportamiento es la clase que se encarga de utilizar frameworks y librerías externas en caso de ser necesitadas.



4.1.1.1 Estructura de la clase

La clase de comportamiento debe seguir la siguiente estructura:

- **Atributos**

La clase debe contener un atributo por cada puerto de salida cuyo nombre debe ser igual al nombre del puerto en el filtro pero quitándole los prefijos del estándar. Dependiendo de la cantidad de datos esperados por puerto, el tipo de datos de cada atributo varía. Si se espera que la funcionalidad del filtro involucre manipular varios valores de entrada en algún puerto, el tipo del atributo debe ser una colección de objetos con tipo de dato igual al del puerto (en el ejemplo de abajo es un ArrayList que contiene objetos de tipo Point3f, el cual es el tipo de dato escogido para la representación de coordenadas 3D). En caso de esperar un solo dato en el puerto, el tipo del atributo debe ser igual al tipo del puerto de entrada.

```
private int output1;

private boolean output2;

private ArrayList<Point3f> outputn;
```

Si se necesitan atributos adicionales para la lógica interna del filtro, se pueden agregar.

- **Constructores**

Se podrán crear tantos constructores como lo requiera el desarrollador dependiendo de la complejidad de la lógica del filtro, sin embargo se deben inicializar los atributos que sean colecciones que representan un número indeterminado de datos de un puerto de salida.

```
private ArrayList<Filter> outputn;

public FilterNameImpl( )

{

    outputn = new ArrayList<Filter>( );
```

```
}
```

En el ejemplo anterior se ilustra un constructor típico.

- **Métodos**

La clase debe tener métodos *get* para todos los atributos. Estos métodos serán utilizados por la parte de integración.

```
private int output1;

private ArrayList<Point3f> outputn;

public int getOutput1( )

{

    return output1;

}

public ArrayList<Filter> getOutputn( )

{

    return outputn;

}
```

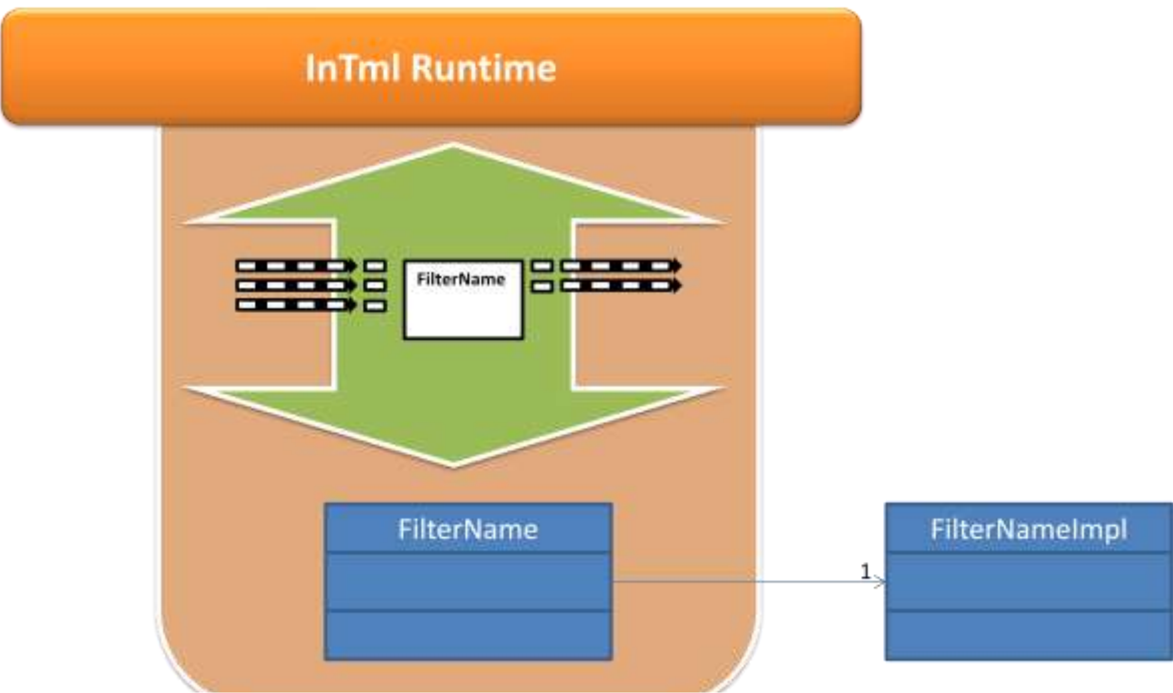
Toda clase de comportamiento de un filtro debe tener el siguiente método:

```
public void execute(int input1, ArrayList<Filter> inputn)

{

    //...

}
```

En este método se ejecuta la lógica del filtro. Los parámetros del método corresponden a los datos que entran por los puertos de entrada al filtro, siguiendo la misma lógica de tipado que se sigue con los atributos de la clase. El método debe poner los datos de puertos de salida en los atributos que los representen después de haber ejecutado la lógica del filtro.

4.1.2 Integración

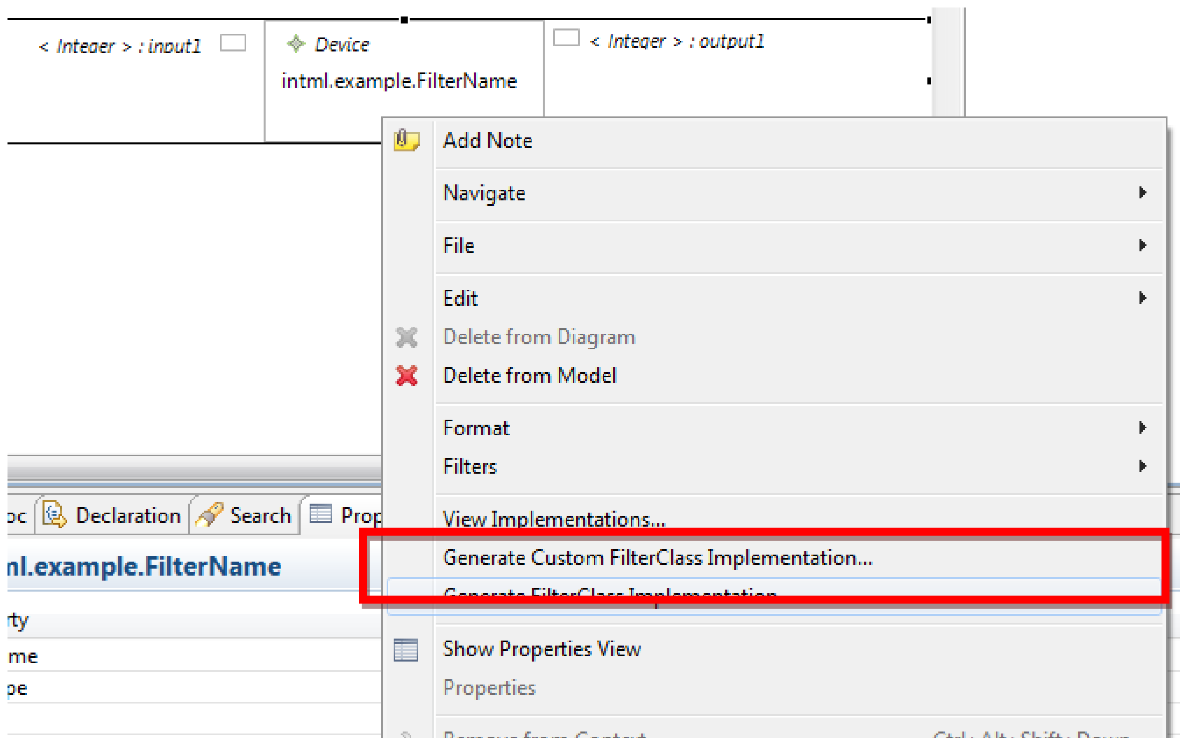
La parte de integración se encarga de integrar la lógica desarrollada en la parte de comportamiento al ambiente de ejecución de InTml. La clase se debe llamar igual al filtro.



Un trabajo importante que debe realizar la parte de integración, es seleccionar de forma determinística un solo valor en los puertos de entrada por donde se espera un solo valor. Esto significa que los parámetros que va a recibir el método *execute(…)* de la clase de comportamiento cuyo tipo no sea una colección, deben pasar por una selección hecha en la parte de integración.

3.1.2.1 Estructura de la clase

Lo primero que se debe hacer es generar la clase con el editor. Para esto, en la librería de filtros donde se encuentra el filtro que se quiere implementar, se hace clic derecho sobre el filtro y se hace clic en *generate FilterClass Implementation…* como se muestra a continuación.



El editor genera la clase de integración con atributos para los puertos, los métodos necesarios, comentarios de donde debería ir el código y demás.

- **Atributos**

Además de los atributos ya generados por el editor, se debe crear un atributo para la clase de comportamiento. El nombre de este atributo puede ser cualquiera.

```
@InTmlPort(intmlType = "Integer", javaType = java.lang.Integer.class)

private AbsPort input1;

/*PROTECTED REGION ID(intml.example.FilterName.Extras) ENABLED START*/

// Extra attributes here

private FilterNameImpl implementation;

// End extra attributes

/*PROTECTED REGION END*/
```

- **Constructor**

El método constructor de la clase debe ser único y es generado por el editor, sin embargo si se desea adicionar lógica al momento de construir el objeto, se puede hacer en la siguiente región:

```
/**
 * Filter SetUp.
 *
*/
public void setup(String name, HashMap<String, String> runtimeProps)
{
      super.setup(name, runtimeProps);
      /*PROTECTED   REGION   ID(intml.example.FilterName.Setup)   ENABLED
      START*/
      // TODO: PutExtra constructor code here
      /*PROTECTED REGION END*/
      this.createPorts();
}
```

Es una buena práctica inicializar el objeto de comportamiento en este segmento para que no se tenga que crear el objeto en cada frame de ejecución, es decir cada vez que se ejecute el método *execute()* del cual se hablará más adelante, de esta forma se inicializa el objeto de comportamiento solo una vez. Sin embargo, si es necesario que el objeto se cree nuevamente en cada frame, también se puede hacer. Un ejemplo de cómo se debería inicializar el objeto de comportamiento es el siguiente:

```
/**
 * Filter SetUp.
 *
 */
public void setup(String name, HashMap<String, String> runtimeProps)
{
      super.setup(name, runtimeProps);
      /*PROTECTED   REGION   ID(intml.example.FilterName.Setup)   ENABLED
      START*/
      this.implementation = new FilterNameImpl( );
      /*PROTECTED REGION END*/
      this.createPorts();
}
```

- **Métodos**

El método más importante es el método *execute( )*, pues es en este método donde se ejecuta la lógica del filtro. En este método se deben seleccionar los valores que van a ser pasados a la clase de comportamiento por medio de los parámetros que se encuentran definidos en el método *execute(…)* de la clase de comportamiento. La forma en que se seleccionan los valores (ya sea en puertos donde se espera un solo valor o donde se esperan varios) es decisión del desarrollador. Después de haber seleccionado los valores a la clase de comportamiento, se debe ejecutar el método *execute(…)* en la clase de comportamiento para que la lógica del filtro sea ejecutada. Una vez ejecutada la lógica, se deben obtener los resultados por medio de los métodos *get* de la clase de comportamiento y deben ser puestos en los puertos de salida del filtro. A continuación se muestra el pseudo-código de una buena forma de desarrollar el método.

```
/**

* Filter execution logic. Process the information collected from the

* IPorts, and send information through the OPorts.

*/

public void execute()

{

      super.execute();

      /*PROTECTED   REGION   ID(intml.example.FilterName.Exec)   ENABLED
      START*/

      Object data;

      while(input1.hasInfo( ))

      {

          data = input1.pop( ).getInfo( );  //Way to get the data

          //Extra data selection logic ...

      }

      implementation.execute( data );  //The parameters passed depends on
      the amount of input ports of the filter.

      // For every value to be sent through the port

      output1.push( new AbsInfo(implementation.getOutput1( )) );

      /*PROTECTED REGION END*/

}
```

Además del método *execute( )* se debe completar una serie de métodos *get* que sirven para obtener el último dato enviado por el filtro en cada puerto de salida. Estos métodos se generan para cada puerto de salida. Dada la forma como está estructurada la implementación, esta tarea puede resultar fácil, pues solo se debe llamar los métodos *get* de cada atributo que represente un puerto de salida en la clase de comportamiento.

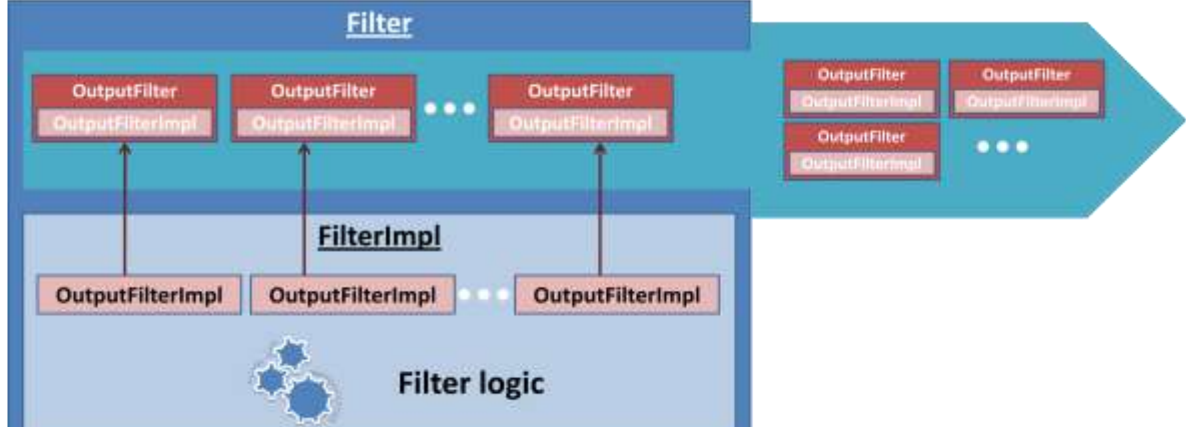
```
//Filter Data Getters.

//Use this GETTERS for accesing the LAST data returned by the Filter.

/**

* Port 'oport.portName' GETTER. Returns the last data procesed.

*/

public java.lang.Integer getOutput1()

{

	/*PROTECTED REGION ID(intml.example.FilterName.output1.Getter) ENABLED START*/

	return implementation.getOutput1( );

	/*PROTECTED REGION END*/

}
```

Si es un puerto de salida por donde se espera arrojar varios valores:

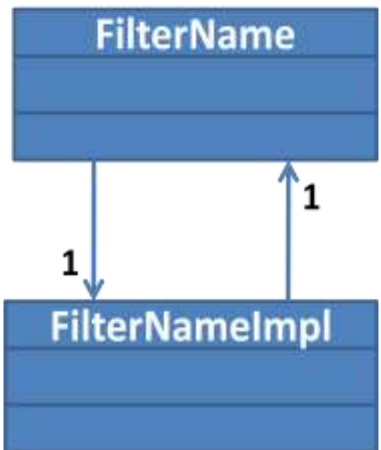
```
		/*PROTECTED REGION ID(intml.example.FilterName.output1.Getter) ENABLED START*/

		int lastIndex = implementation.getOutput1( ).size( ) - 1;

		return implementation.getOutput1( ).get(lastIndex);

		/*PROTECTED REGION END*/

	}
```

4.1.3 Dependencia circular

De la forma en que está definida la estructura de cada filtro, la clase de integración conoce la clase de comportamiento pero no viceversa. En ocasiones es necesario que la clase de comportamiento conozca a la clase de implementación para la identificación de los filtros dentro del ambiente de ejecución. Este caso se presenta cuando se tiene un filtro que tiene un puerto de salida cuyo tipo de dato es otro filtro (no un tipo básico de InTml como Pos3D, Boolean, Float, etc.). En la figura de abajo se observa como la lógica del filtro se ejecuta utilizando las clases de comportamiento de las instancias de los filtros que se deben arrojar por el puerto, pero sigue siendo necesario enviar la clase de integración por el puerto.



Es por esto que la clase de comportamiento necesita conocer a la clase de integración, pues es la forma de enviar el filtro completo introduciendo la menor cantidad de dependencias. Entonces, el diagrama UML de un filtro que deba ser enviado por un puerto de otro filtro es el siguiente.



4.2 Descripción especifica de la implementación de cada filtro

En esta sección se documentará los detalles de la implementación de los filtros descritos en la sección de especificación. Se presentarán los detalles de cada filtro en el mismo orden en que aparecen en la sección de especificación.

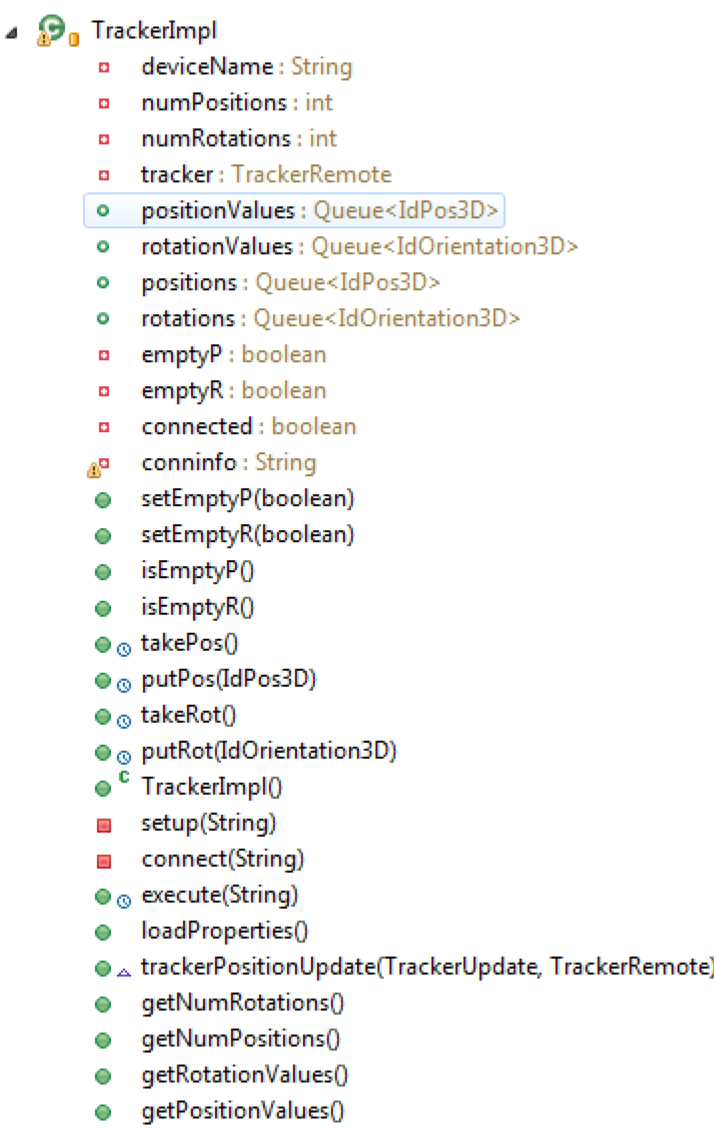
Tracker

Los datos enviados por el dispositivo se reciben utilizando VRPN, por lo tanto, lo primero que se debe verificar en el método execute es la conexión con el servidor VRPN. Para esto se utiliza un atributo de tipo booleano que cumple con la función de bandera indicando el estado (conectado o no) de la conexión VRPN. Si no se encuentra conectado al servidor, se ejecuta un método para conectarse. Cada vez que se envían datos desde el dispositivo, se ejecuta un método llamado trackerPositionUpdate el cual mete dicha información en dos colas de datos: una para posición y otra para orientación. En el método execute, luego de haber verificado la conexión, se ponen los datos de la cola en el atributo que representa el puerto de salida, esto se hace tanto para datos de

posición como de orientación. Para finalizar, la cola de datos se crea de nuevo para que quede vacía, lista para esperar datos provenientes del dispositivo. Como los datos del dispositivo provienen de un hilo de ejecución diferente al principal de InTml, y por lo tanto los datos de la cola son llenados por un hilo de ejecución diferente, el método execute de la clase de comportamiento es un método sincronizado para evitar problemas por accesos concurrentes a la cola.

El número de datos que arroja el dispositivo (orientación y posición), es un dato cargado desde un archivo de propiedades, y se espera que haya un archivo de propiedades por cada dispositivo soportado.

Listado de métodos y atributos de la clase de comportamiento



TrackerSelector

Para hacer la selección de valores, se debe tener el identificador que va a indicar los valores de posición y orientación que se desean. Lo primero que hace el filtro es un chequeo sobre estos

valores para ver si entró un valor nuevo. Si entró un valor nuevo, la selección se hace usando este y se guarda para futuras selecciones, si no entró ningún valor, la selección se hace con el último valor que entró. Posteriormente se hace un recorrido por los valores de posición y orientación enviados por el Tracker y se selecciona el ultimo valor que contenga un identificador igual al que se esté utilizando en ese frame.

Listado de métodos y atributos de la clase de comportamiento

- TrackerSelectorImpl
  - desPosInternal : Integer
  - desOrientInternal : Integer
  - position : Point3f
  - orientation : Quat4f
  - TrackerSelectorImpl()
  - getPosition()
  - getOrientation()
  - execute(Integer, Integer, ArrayList<IdPos3D>, ArrayList<IdOrientation3D>)

OffSetter

Lo primero que hace el filtro es guardar los valores de posición, orientación y escalamiento con los cuales se va a modificar la orientación y posición que entran por los puertos “s_in_position” y “s_in_orientation”. De esta forma se asegura que las transformaciones se van a realizar con los últimos valores que entraron. La posición guardada se suma en cada dimensión con la posición que debe ser modificada y luego se aplica un escalamiento al resultado obtenido. La orientación que se guardó se normaliza para asegurar resultados normalizados y luego se multiplica el cuaternión guardado con el cuaternión de la orientación que debe ser modificada.

Listado de métodos y atributos de la clase de comportamiento

- OffSetterImpl
  - positionOffset : Point3f
  - orientationOffset : Quat4f
  - scaleEstado : Float
  - position : Point3f
  - orientation : Quat4f
  - OffSetterImpl()
  - getPosition()
  - getOrientation()
  - execute(Point3f, Quat4f, Point3f, Quat4f, Float)

PosToPosOrient

Al igual que en los filtros anteriores, los valores de entrada se guardan y se utilizan los últimos valores que le entraron al filtro para realizar los cálculos. Es decir, las dos posiciones de entrada se guardan en el filtro y luego se procede a realizar los cálculos. La posición de salida es igual a la posición 1. Luego se procede a calcular el cuaternión de la orientación, el cual requiere varios pasos. Lo primero que se hace es calcular el vector que representa la orientación de las dos posiciones, y se hace restando la posición 1 a la posición 2 (posición 2 – posición 1). Luego se toma el vector [0, 1, 0], el cual representa la orientación de una matriz de transformación inicial, y

se calcula el producto cruz de ambos vectores. Después se calcula el ángulo entre ambos vectores. Finalmente se crea una representación de ángulo eje para la orientación (tomando el vector resultado del producto cruz y el ángulo entre los dos vectores) y luego se calcula el cuaternión correspondiente a dicha representación.

Listado de métodos y atributos de la clase de comportamiento

- PosToPosOrientImpl
  - pos1Temp : Vector3f
  - pos2Temp : Vector3f
  - position : Point3f
  - orientation : Quat4f
  - PosToPosOrientImpl()
  - execute(Point3f, Point3f)
  - getPosition()
  - getOrientation()

RayCaster

Para obtener los objetos que están siendo intersectados por el rayo, se utiliza el método pickAll(…) de Java3D. En el caso en que la instancia del filtro Ray que se esté utilizando, utilice una longitud finita, se calcula el punto final del rayo utilizando los datos de posición, orientación y longitud que tenga la instancia de Ray. Si no usa una longitud finita, se debe calcular un vector de dirección utilizando la orientación. Después se utiliza el método pickAll(…) utilizando como parámetros la posición del rayo como punto de origen y el vector de orientación para el caso de no utilizar longitud finita. Si se usa longitud finita los parámetros son la posición del rayo como punto de origen y el punto final que se calculó. Los resultados que devuelve Java3D son los nodos en el grafo de escena que representen algún tipo de geometría, es por eso que se usa el concepto de UserData en todos los nodos de geometría para los objetos. El UserData permite asociar cualquier tipo de objeto Java a cualquier nodo del grafo de escena, en este caso se utiliza para asociar la clase de comportamiento a los nodos de geometría. Para asegurar que no hayan objetos repetidos, se hace un recorrido sobre la UserData de los nodos de geometría intersectados por el rayo para chequear que no se encuentren repetidos y que sean objetos seleccionables.

Listado de métodos y atributos de la clase de comportamiento

- RayCasterImpl
  - selectedObjects : ArrayList<SimpleObjectImpl>
  - RayCasterImpl()
  - execute(RayImpl, ArrayList<SimpleObjectImpl>)
  - getSelectedObjects()

FeedbackToggle

Primero se guarda el valor de estilo que se va a utilizar para que siempre se utilice el último valor de estilo que ingreso al filtro. Luego, si la bandera para activar el cambio de apariencia está habilitada, se procede cambiar la apariencia de los objetos según el estilo seleccionado y se guardan como objetos previos. Finalmente los objetos anteriores retornan a su apariencia normal.

Listado de métodos y atributos de la clase de comportamiento

```
FeedBackToggleImpl
    STYLE_MATERIAL : String
    STYLE_BOUNDING_BOX : String
    style : String
    previousObjects : Set<BasicObject>
    FeedBackToggleImpl()
    execute(ArrayList<BasicObject>, boolean, String)
```

ObjectSelector

Primero se guarda el grupo de identificadores de objetos que deben ser utilizados para la selección de objetos. Luego se hace un recorrido por los objetos que entran y si hay algún identificador guardado que sea igual al identificador del objeto, el objeto se clasifica como objeto seleccionado, de lo contrario se clasifica como objeto no seleccionado.

Listado de métodos y atributos de la clase de comportamiento

```
ObjectSelectorImpl
    objectIdsInternal : ArrayList<String>
    selectedObjects : ArrayList<SimpleObjectImpl>
    nonSelectedObjects : ArrayList<SimpleObjectImpl>
    ObjectSelectorImpl()
    getSelectedObjects()
    getNonSelectedObjects()
    execute(ArrayList<SimpleObjectImpl>, ArrayList<String>)
```

Scene

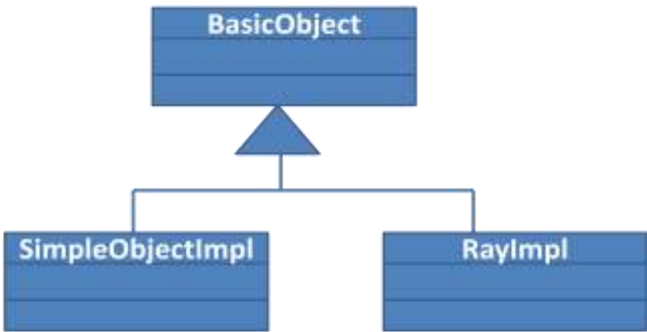
El filtro cuenta con una bandera interna que indica si la escena ya fue cargada o no. Por lo tanto primero se verifica esa bandera. Si no ha sido cargada, se utiliza un API de Java3D que sirve para cargar archivos VRML dentro del grafo de escena. Se utiliza el API para recorrer los nodos que contengan una cadena dentro del campo DEF. Si estos nodos contienen el prefijo “Object_”, dentro del campo DEF entonces el objeto se crea como una instancia de SimpleObject y se carga detro de la escena.

Listado de métodos y atributos de la clase de comportamiento

```
SceneImpl
    loaded : boolean
    sceneGraph : BranchGroup
    objectStream : ArrayList<SimpleObjectImpl>
    SceneImpl(BranchGroup)
    execute(String)
    getObjectStream()
```

SimpleObject

Dado que este filtro y Ray tienen muchas características similares, se decidió crear una clase abstracta que reúne las características comunes de ambos filtros. Esta clase se llama BasicObject. Por lo tanto, la clase de comportamiento de SimpleObject y de Ray extienden de esta clase.



SimpleObjectImpl cuenta con dos constructores: uno para cargar un objeto desde el filtro Scene y otro para cargar el objeto como un filtro en un diagrama de aplicación. En el método execute se llama un método de BasicObject llamado setParameters(…) el cual se encarga de actualizar los parámetros de posición, orientación y demás atributos que tengan que ver con los puertos de entrada comunes a ambos filtros. Posteriormente se ejecuta un el método computeOutputs que también se encuentra en BasicObject cuyo propósito es fijar las salidas del filtro en los atributos correspondientes.

Listado de métodos y atributos de la clase de comportamiento

- SimpleObjectImpl
  - filter : SimpleObject
  - SimpleObjectImpl(BranchGroup)
  - SimpleObjectImpl(TransformGroup, String, BranchGroup)
  - SimpleObjectImpl(BranchGroup, SimpleObject)
  - execute(String, Point3f, Quat4f, float, boolean, boolean, Filter, boolean, BasicObject, boolean, boolean)
  - getFilter()

Listado de métodos y atributos de la clase BasicObject

- BasicObject
  - selectable : boolean
  - sgParent : Group
  - objectBranchGroup : BranchGroup
  - group : TransformGroup
  - isVisible : boolean
  - bBoxBranch : BranchGroup
  - bBox : TransformGroup
  - branchGroup : BranchGroup
  - position : Point3f
  - orientation : Quat4f
  - scale : float
  - parent : BasicObject
  - id : String
  - BasicObject(BranchGroup)
  - getGroup()
  - getPosition()
  - getOrientation()
  - getScale()
  - isSelectable()
  - getParent()
  - getId()
  - setupObjectFromFile(String)
  - setUserDataFromHere(Node)
  - setParameters(Point3f, Quat4f, float, boolean, boolean, BasicObject, boolean, boolean)
  - showBBox()
  - clearBBox()
  - computeOutputs()
  - listShapes(ArrayList<Shape3D>, Enumeration)

Ray

A diferencia de SimpleObject, la clase de comportamiento de Ray solo tiene un constructor utilizado para usar Ray en un diagrama de aplicación. En el método execute se realizan las acciones que se hacen en SimpleObject y se manejan los datos relacionados con la longitud y el radio del rayo. Adicionalmente, si no se encuentra una ruta para cargar el objeto que representa al rayo, se crea un cono como geometría a utilizar por defecto.

Listado de métodos y atributos de la clase de comportamiento

- RayImpl
  - useLenghtForPicking : boolean
  - filter : Ray
  - lenght : float
  - radius : float
  - RayImpl(BranchGroup, Ray)
  - isUseLenghtForPicking()
  - getLenght()
  - getRadius()
  - execute(String, Point3f, Quat4f, float, boolean, boolean, Filter, boolean, BasicObject, boolean, boolean, boolean, float, float)
  - getFilter()