UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5

Sistema de Replay para aplicaciones de Realidad Virtual

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Autores: Yeisel López Rodas

Felipe Pérez Hernández

Tutores: Ing. Osley Bretau Camejo

Ing. Raissel Ramírez Orozco

Ciudad de la Habana

Junio de 2009

Datos de Contacto:

Nombre y Apellidos: Osley Bretau Camejo

Edad: 25 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Profesor Adiestrado

E-mail: obretau@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 5 años de experiencia en proyectos de Realidad Virtual.

Nombre y Apellidos: Raissel Ramírez Orozco

Edad: 25 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Profesor Adiestrado

E-mail: rramirez@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 5 años de experiencia en proyectos de Realidad Virtual.

Dedicatoria:

…A mis padres Felipe y María

A mis abuelos Sara, Eugenia, Pablo y Bernabé

A mi novia Beatriz.

Felipe

…A mis padres Modesto y Consuelo

A mis hermanos Yaiselin y Yeniel

A mi novia Liset

…ustedes son mi razón de ser…

Yeisel

Agradecimientos:

El tener que escribir estas líneas significa que ha llegado el momento final de un largo período de tiempo que comenzó en la escuelita rural de un lugar casi desconocido donde se promediaba alrededor de 5 estudiantes por grado y que está culminando en una gigante y maravillosa universidad de más de 10 000 estudiantes. Por ser este el resultado de un proceso donde han influido muchas personas durante bastante tiempo quisiera agradecer de todo corazón a todos los que han tenido que ver de una forma u otra con mi educación, lo cual me ha permitido llegar hasta aquí. Finalmente quiero agradecer con especial atención a los que me han brindado su ayuda para lograr realizar este trabajo, a los cuales les estaré eternamente agradecido.

A mis padres, Felipe y María, por su amor y apoyo, han sido mi principal fuente de inspiración en todo lo que he hecho.

A mi novia Beatriz, su ayuda y amor me han facilitado mucho las cosas.

A nuestros tutores Raissel y Osley, por confiar en nosotros, ayudarnos y más que buenos tutores haberse convertido en buenos amigos.

A Ernesto Carrasco por su amistad e incondicional ayuda.

A nuestros profes Ernesto de la Cruz, Villanueva, Osvaldo, Oliver, por su ayuda cuando los hemos necesitado.

A Duniel del Pino, por su gran ayuda en la confección de la presentación final.

A los amigos del apartamento y de Colón, por su amistad y por haber estado al tanto de nuestro trabajo, Rubencito, Pelusa, Lizardo, Yadiel, Dismel, Pedrito, el Beto, Joan, Iriam….

A mis suegros Jorge e Iliana, por haberse comportado como dos padres más para mí.

A Yeisel, quien además de compañero de tesis y amigo se ha convertido en un hermano para mí.

Agradecimientos:

A mis padres por toda la confianza depositada en mí, por enseñarme a luchar por mis sueños, este triunfo mío también es suyo.

A ti Papi te dedico este mi más grande esfuerzo por ayudarme a escalar cada peldaño en la vida, por enseñarme tantas cosas y por apoyarme cada día que he pasado lejos de ti.

A ti Mami, tú que con más sufrimiento que alegrías me apoyaste en este camino que decidí tomar, te dedico todos mis logros y en especial este el más grande, por tenerme siempre presente, por no olvidar nunca cada detalle de mi vida por habérmela dado y por ser lo más tierno y dulce que he llegado a conocer.

A mi hermana por enseñarme el camino, por ser mi fuente de inspiración y mi ejemplo a seguir.

A mi hermano por servirme de inspiración para ser mejor cada día con tal de que siga mi ejemplo y acepte mi apoyo cuando lo necesite.

A mis abuelos, los mejores del mundo, no hay una sola línea de este trabajo donde no estén presentes. Vivan muchos años más; que aún los necesito mucho.

A mi Tía Katy, Tía Carmen, Tío Joel, a mis primos Mirocha y Jorgito este trabajo no hubiese sido posible sin su apoyo incondicional, sus frases de amor y fuerza, su cariño infinito que los han hecho parte importante de mi vida.

A mi casi hermana Zenia, gracias por estar ahí cuando te necesite, por enseñarme el significado de la palabra “amistad”.

A mi casi hermano Yunier, a pesar de la distancia estos agradecimientos no tendrían sentido si tú no estás presente.

A Gladys y Eduardo gracias por estar dispuestos a brindarme su apoyo cuando lo necesite.

A todos mis amigos de la UCI, que por ser muchos no me atrevo a mencionar nombres por miedo a que se me olvide algo, gracias por contar con su apoyo cuando lo necesité y sus críticas constructivas que me ayudaron a ser mejor cada día, no los voy a olvidar por hacer más placentera mi estancia aquí y marcar una etapa en mi vida.

A todos los profesores que durante todos estos años contribuyeron con mi formación profesional.

A los que leyeron mi tesis porque con sus comentarios y observaciones me permitieron enriquecerla.

A mis tutores Raissel y Osley, gracias por su alto rigor científico, por el tiempo y el esfuerzo que han puesto en la realización de este trabajo.

A mi compañero de Tesis y hermano Felipe, ha sido un placer compartir contigo estos años de universidad y las malas noches que empleamos en la realización de este proyecto por fin te puedo decir: …al fin lo logramos!!!

A mi novia Liset por ser la persona más paciente que he tenido a mi lado, por estar siempre dispuesta, por no abandonar esta larga espera, por sensibilizarse con todo lo que me concierne, por todo su amor y entrega ¿Qué sería de mi si no fuera por ti?

Resumen

Los SRV han alcanzado un impresionante desarrollo, su aplicación es de vital importancia en temas como el entrenamiento de diferentes actividades humanas y el ocio. Un importante requerimiento en la mayoría de estos sistemas es su capacidad de efectuar el proceso de repetición (*Replay*). El objetivo de este trabajo es proponer una solución con la funcionalidad de repetición para SRV.

Se ha desarrollado un estudio del estado del arte a nivel mundial de los principales enfoques utilizados con este propósito, los cuales son fundamentalmente, el conocido como enfoque Determinista y el nombrado Salvar Estados.

Como resultado final se propone la implementación del enfoque determinista, asociada al motor gráfico OGRE 3D (*Object-Oriented Graphics Render Engine*), guiado por el proceso unificado de desarrollo (RUP) e implementado en C++ estándar.

Palabras Clave

Proceso de repetición, enfoques, determinista, salvar estados.

ABSTRACT

Virtual reality systems have achieved a stunning development; its implementation is of crucial importance on topics such as training of various human activities and entertainment. An important requirement in most of these systems is their ability to repeat the process (Replay). The objective of this work is to propose a solution with the functionality of replication for virtual reality systems.

It has been developed a state of the art of main approaches used for this purpose, which is essentially the approach known as deterministic and the one known as Save States.

The end result was the implementation of the proposed approach, associated with the OGRE 3D Engine graphics (Open Graphics Render Engine), guided by the unified process of development (RUP) and implemented in C + + standard.

Keywords

Approach, Virtual Reality, System of repetition (replay).

Contenido

Índice de Figuras

Índice de Tablas

Introducción

Durante las últimas décadas el mundo ha evidenciado el más vertiginoso desarrollo que cualquier ciencia haya jamás alcanzado en tan poco tiempo. La informática se ha difundido y evolucionado de manera impresionante, al punto de que cada día aparecen nuevas tecnologías con disímiles funcionalidades que potencian el desarrollo de muchas de sus áreas a la vez que contribuyen con un número cada vez mayor de procesos sociales y científicos. Los avances obtenidos en materia de visualización y modelación gráfica han hecho que muchos desarrolladores de *software* y *hardware* enfoquen sus metas hacia la Realidad Virtual (RV), campo que propone soluciones a disímiles problemáticas de la vida cotidiana. Actualmente existen resultados inspiradores en esta área, es el caso de los muy difundidos videojuegos o simuladores.

No es extraño encontrarse hoy en día Sistemas de Realidad Virtual (SRV) con un elevado nivel de realismo, donde se simulan escenarios reales, objetos, personas, e incluso eventos de difícil representación como la lluvia, el viento, las luces y los movimientos faciales, todos con niveles de detalles extremadamente altos.

Los simuladores constituyen parte importante en el desarrollo actual de otras ciencias debido a que proponen sistemas seguros para el aprendizaje, para la prevención de fenómenos o catástrofes y el estudio de complejos sistemas de la vida real garantizando la disminución de errores a partir de la adquisición de experiencia.

Diferentes ramas han adoptado los simuladores para ganar en exactitud, ejemplo de ello son la medicina y el sector militar. Simular en un entorno virtual actividades específicas y complejos procesos de la vida real que requieren conocimiento y práctica; que en determinados casos implican gastos astronómicos además de riesgos significativos para la integridad física e incluso hasta la vida de personas es una innegable ventaja. Los simuladores han revolucionado los métodos de formación, enseñanza y experimentación, desplazando muchas veces técnicas tradicionales. Sin embargo, a pesar de que se han desarrollado simuladores con grandes

potencialidades y niveles de realismo asombrosos aún se trabaja en su perfeccionamiento en aras de ampliar sus funcionalidades y alcance.

Cuba, ha encontrado en la RV una eficaz vía de solución a numerosos problemas. En el epicentro de este proceso está el Polo de Realidad Virtual (PRV) de la Facultad 5 (F5) en la Universidad de las Ciencias Informáticas (UCI). El trabajo de este polo ha sido enfocado hacia la investigación y producción de sistemas de este tipo con la intención de situarlos a la altura de los requerimientos actuales de estos sistemas en el marco internacional.

A pesar de que los resultados obtenidos hasta el momento en los proyectos productivos del PRV son alentadores solo constituyen la primera etapa de un largo y complejo proceso hacia la excelencia que requiere el mercado del *software*. Constantemente han surgido nuevas propuestas, que con el paso del tiempo han ido siendo más ambiciosas y a la vez necesitando funcionalidades muy específicas, es el caso de videojuegos y otros SRV que requieren en algunas circunstancias de la posibilidad de repetir procedimientos o jugadas antes realizadas.

En diversas circunstancias se **necesita** reproducir una secuencia de acciones que tuvieron lugar en una escena determinada mediante la grabación de las mismas. Debido a la inexistencia de esta funcionalidad, en el PRV los usuarios no pueden grabar ejercicios antes ejecutados para su futuro análisis, revisión, y evaluación por parte de especialistas, como tampoco los usuarios involucrados podrían observar una ejecución previa que le permita identificar errores y adquirir experiencia para otras interacciones.

Partiendo de la usabilidad y ventajas que proporcionan los Sistemas de Repetición y el gran número de proyectos del PRV que demandan su uso se puede concluir su importancia. Después de analizar e identificar la situación existente este trabajo pretende brindar una solución al siguiente **problema científico**: Inexistencia de la funcionalidad de *replay* en las aplicaciones del PRV. Por lo que en este trabajo se propone como **objetivo general:** implementar un sistema que permita lograr la funcionalidad de *replay* en SRV. El **objeto de estudio** de este trabajo es el tratamiento de objetos virtuales en motores gráficos y SRV, siendo el **campo de acción** las técnicas utilizadas para desarrollar la funcionalidad de *replay*

en SRV Para dar cumplimiento a los objetivos planteados se trazan las siguientes **tareas investigativas**

1. Búsqueda de información de las técnicas existentes en el mundo sobre desarrollo de *replay*, para la elaboración del estado del arte de la investigación.
2. Realización de un estudio de los diferentes motores gráficos y de videojuegos para entender su funcionamiento.
3. Caracterización de las diferentes técnicas de implementación de un *replay*, para seleccionar la más adecuada a la solución.
4. Implementar una versión inicial de un sistema de *replay* para dar solución al problema planteado.

Para una mejor comprensión, el documento está dividido en capítulos que estructuran el contenido de la siguiente forma:

En el **capítulo 1** “Fundamentación Teórica”, se hace un análisis del estado del arte de los diferentes enfoques existentes para crear un sistema de *replay*, destacando ventajas y desventajas de cada una con el objetivo de esclarecer cual es posible candidato para una solución.

El **capítulo 2** “Soluciones Técnicas”, establece la solución resultante del análisis realizado en el capítulo anterior y se detalla el enfoque a utilizar finalmente.

En el **capítulo 3** “Descripción de la Solución Propuesta”, se describe el sistema a desarrollar desde la perspectiva de las necesidades del cliente, en función de las dificultades, necesidades y características del mismo, y aplicando las técnicas a utilizar descritas en el capítulo anterior.

El **capítulo 4** “Diseño del Sistema”, corresponde a los flujos de trabajo de análisis e implementación de RUP, se describen las clases de diseño, se relacionan mediante el diagrama de clases de análisis y se distribuyen en componentes de *software* como indica el diagrama de componentes del sistema.

Capítulo 1: Fundamentación Teórica

Introducción:

Es común observar en la vida cotidiana a través de la televisión, principalmente en programas deportivos, como se pueden repetir escenas de algo sucedido en un determinado momento, además de una precisión exacta a como se produjo originalmente.

En los SRV, especialmente en simuladores y videojuegos, este mismo efecto se denomina *replay*. Para lograrlo es necesario el conocimiento de un importante grupo de elementos, que influyen a la hora de poder desarrollar con éxito esta funcionalidad en los SRV.

En el presente capítulo, se presenta un estudio del estado del arte relacionado con el desarrollo de *replay*, para ello se presentan una serie de conceptos y temas relacionados, distribuidos en diferentes secciones. Por último se hace referencia a diferentes enfoques, utilizados por los desarrolladores de estos sistemas a nivel mundial, para dar solución a este problema.

1.1 Visión General

Los *replay* tienen gran aceptación y utilidad en las aplicaciones gráficas interactivas, especialmente en los videojuegos y simuladores por sus característica de mostrar nuevamente los eventos ya sucedidos, pudiendo ser detallados con mayor precisión y cambiando a diferentes vistas todos los gráficos del sistema, incluso los que no son vistos por el usuario en el momento de la ejecución por la perspectiva de la cámara en ese momento.

Para lograr la ejecución exitosa de un *replay* es necesario tener un control total del comportamiento e interacción de todos los objetos presentes en la escena del entorno virtual, tarea que no es tan sencilla si se tiene en cuenta la diversidad de objetos que pueden existir en una escena, así como de todos los factores que pueden influir en el comportamiento de los mismos. Para todo esto es necesario tener conocimiento absoluto del comportamiento del *software* utilizado para la construcción del sistema, prestándole especial atención a la estructura de datos empleada para controlar los objetos presentes en la escena, así como también se debe tener conocimiento sobre el comportamiento de algunos componentes del *hardware* que pueden influir en el exitoso desarrollo del *replay*.

1.2 Breve introducción a la realidad virtual. Características básicas de un sistema de realidad virtual

La Realidad Virtual es una tecnología enfocada hacia la simulación de situaciones, objetos, actividades, entornos y fenómenos del mundo real, con un objetivo cognitivo. La misma ha tomado un gran auge sobre todo en las últimas tres décadas. En áreas como los videojuegos y simuladores se puede observar su amplio desarrollo, como también por su difusión a través de internet donde ahora se puede escuchar hablar de temas como entidades virtuales, aulas virtuales, firmas virtuales, amigos virtuales y hasta matrimonios virtuales. Aunque en los últimos tiempos ha alcanzado su máximo desarrollo, el término de RV tiene su origen en la década del 60, al ser introducido su concepto en un artículo escrito por Ivan Sutherland, publicado en el año 1965 bajo el título de *"The ultímate display"* *[14]* , el cual fue de gran ayuda para el desarrollo posterior de esta tecnología, la cual puede definirse como un espacio tridimensional gráfico, táctil, visual o auditivo (también llamado ambiente o "mundo" virtual) generado por computadora, donde los usuarios pueden interactuar y navegar en ese espacio utilizando equipos especiales [1].Y en la cual no se lograron avances significativos hasta los años 80, ya que hasta ese momento las computadoras personales no eran lo suficientemente poderosas en cuanto a despliegue gráfico, memoria y capacidad de procesamiento [2].

1.2.1 Características Básicas de un SRV

Las tres características básicas de un SRV son la interacción, inmersión y la tridimensionalidad.

Interacción: La interacción es quien permite al usuario manipular el curso de las acciones que se llevan a cabo dentro de una aplicación de realidad virtual, con el propósito que el sistema responda a los estímulos del usuario. Esta característica posee dos aspectos importantes:

- Navegación: Son los mecanismos que tiene el sistema para brindarle al usuario libertad de movimiento dentro del mundo virtual, al permitirle caminar, nadar, volar, correr, etc. Este aspecto comprende también, la factibilidad que tiene el SRV para que

el usuario pueda ver el mundo con el cual está interactuando desde varios puntos de vista.

- Dinámica del ambiente: Son las reglas que posee el sistema para condicionar cómo los componentes del mundo virtual van a interactuar con el usuario para intercambiar energía o información.

Inmersión: La inmersión es la capacidad del sistema para bloquear toda distracción del usuario y enfocarlo selectivamente solo en la información u operación sobre la cual está trabajando. Esta característica 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).

Tridimensionalidad: La tridimensionalidad tiene que ver directamente con la manipulación de los sentidos del usuario, principalmente la visión, para dar forma al espacio virtual; 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) [2].

1.3 Motores Gráficos

Un motor grafico es un *software* que acepta comandos de aplicaciones y construye imágenes y textos que son dirigidos a los driver gráficos y al *hardware* gráfico[15]. Es la parte de un programa que controla, gestiona y actualiza los gráficos 3D en tiempo real[16].

Los motores gráficos tienen la función de cargar, visualizar, manipular y administrar todos los datos relativos al contenido gráfico y los efectos visuales. Modelos 3D de objetos, paisajes, edificios, animales y jugadores pueden ser cargados, texturizados, iluminados y animados. Efectos adicionales (tales como niebla, distorsión del lente …) pueden ser adicionados para enriquecer el realismo visual [17].

Usualmente el motor gráfico no es suficiente, en muchos casos las aplicaciones requieren además de sonido, inteligencia artificial, comportamiento físico realista, transmisión de datos por red entre otras funcionalidades en dependencia del propósito. Las herramientas para este fin pueden ser creadas por el propio equipo de desarrollo, pero lo más común es que se utilice alguna de las existentes, esto es lo más práctico en la mayoría de los casos, se pueden encontrar tanto comerciales como de código libre, los motores gráficos de gran calidad en caso de ser comerciales pueden tener precios elevados. En el caso de desarrollar su propio motor gráfico debe estar a la par del *hardware* gráfico disponible, ya que uno de los principales problemas del desarrollo de los mismos es que tratando de buscar un mayor realismo, se convierten en sistemas muy complejos computacionalmente por lo que pierden la interactividad. A continuación se mostrará el ejemplo de cómo se desarrolla el trabajo con los objetos y su representación en escena a través del grafo de escena de dos motores gráficos, el conocido con el nombre de *SceneToolKit* (STK) y el segundo nombrado OGRE 3D.

1.3.1 SceneToolKit

STK es la primera y una de las herramientas brindadas por el Proyecto de Herramientas de Desarrollo para SRV de la F5 de la UCI, teniendo como objetivo básico agrupar las funcionalidades comunes a cualquier SRV, de manera que se les facilite el trabajo a los programadores gráficos a través de la reutilización de código.
Esta herramienta, en desarrollo actualmente, permite no solamente la visualización de los entornos sintéticos sino además la aplicación de leyes físicas y matemáticas, animaciones, etc., usando las bibliotecas gráficas OpenGL y DirectX, sobre plataforma Windows y Linux. A la vez, se retroalimenta con las necesidades de los programadores que las utilicen, así como de sus investigaciones [3].

Características Generales del Trabajo con los Objetos y Control de la Escena

STK permite manipular geometrías (polilíneas y mallas triangulares), luces, cámaras, personajes y componentes visuales (botones, paneles...), así como los nodos del grafo de escena.

De los objetos y propiedades que se pueden manipular durante una simulación o juego, algunos merecen ser identificados por nombres o "*id*", mientras que en otros casos esto es innecesario. Existen dos clases bases en STK de las cuales heredan todos los objetos ya sea directa o indirectamente, una clase que se encarga de representar todos los objetos con nombre e identificador y la otra de la cual heredan directa o indirectamente todos los objetos de la STK.

Los ID de los objetos que lo requieran se generan durante la carga, o más bien en el momento de creación de los objetos, y de forma autonumérica, independientemente del ID que dichos objetos tengan en el entorno de diseño donde hayan sido creados.
A su vez, los nombres por defecto se crean como "*Name* + ID", por ejemplo, un objeto con ID = 239 tendrá el nombre "*Name239*".

En el caso de que los objetos tengan un nombre desde el fichero de donde se carguen, se les pone dicho nombre, y además se brinda la opción de entrar un nuevo nombre por parte de los usuarios. Por tanto los nombres de los objetos no son únicos y no se utilizan como identificadores de los mismos, aunque están implementados métodos de búsqueda tanto por ID como por nombre, pero la herramienta no se responsabiliza de diferenciar los objetos por su nombre.

Por último, existen entidades que contienen a otras, como es el caso de los nodos del grafo de escena, que son estructuras que contienen, por ejemplo, geometrías. Dichos nodos también tienen un ID único, pero por defecto toman el nombre del objeto que contengan.

Grafo de Escena

En la STK se controlan los objetos de la escena a través de un grafo, de manera que ningún objeto en sí (dígase geometrías) es dibujado, a no ser que sea insertado en dicha estructura. Es decir, cada objeto que se cree debe ser insertado en el grafo de escena para poderlo controlar y visualizar.

Los grafos de escenas son estructuras que admiten varias subestructuras, de forma que puede contener ramas que sean *Quadtrees*, *BSP trees*, etc. Además, una modificación espacial de los objetos de la escena, por ejemplo, una traslación, no conlleva a reformar la estructura del grafo, sino que se cambian los datos del estado geométrico en el nodo correspondiente, lo que lo hace óptimo para escenarios con muchos objetos dinámicos.

En STK los objetos son almacenados en una lista única, y en el grafo de escena se pueden contener varias referencias a ellos, de esta manera, por ejemplo, se pueden tener varias instancias de un mismo árbol por toda la escena almacenando a éste en una sola dirección de memoria, sólo que se representa con varios estados geométricos y de *render* en diferentes lugares. La estructura básica del grafo es el nodo. Los nodos utilizados por la STK para la construcción del grafo de escena utilizándolos para determinadas necesidades son:

- Nodo base.
- Nodo agrupador.
- Nodo geometría.
- Nodo cámara.
- Nodo luz.
- Nodo esqueleto.
- Nodo hueso.
- Nodo para animación de texturas.
- Nodo rejilla.
- Nodo Quadtree de terrenos [3].

1.3.2 Ogre

Ogre es un flexible motor 3D orientado a escena, escrito en C++ y diseñado para dar facilidades a desarrolladores que producen aplicaciones que utilizan aceleradores gráficos 3D de *hardware*. La biblioteca de clases resume todos los detalles del uso de las bibliotecas subyacentes del sistema como Direct3D y OpenGL y provee una interfaz basada en los objetos mundiales y otras clases intuitivas [13].

Características Generales del Trabajo con los Objetos y Control de la Escena

Ogre permite manipular todo tipo de objetos tanto dinámicos como estáticos: luces, cámaras, sistemas de partículas, entidades, nodos, entre otros.
Las entidades componen todos los objetos rendereables de la escena y están compuestos por mallas y sus materiales asociados, los nodos de escena son utilizados para ubicar todos los objetos dentro del mundo virtual y pueden tener atachados: entidades, cámaras, luces, nodos hijos, entre otros. Tanto las entidades como los nodos necesitan tener un nombre único como identificador. Existen dos clases bases en Ogre de las cuales heredan todos los objetos ya sea directa o indirectamente.

Administrador de Recursos

Todo lo que Ogre necesita para el *render* de una escena es conocido como recurso y todos los recursos son manejados por el objeto *ResourceGroupManager*. Este objeto es responsable de localizar e inicializar los tipos de recursos que conoce.
Ogre reconoce los siguientes tipos de recursos:

- Mallas.
- Esqueletos.
- Materiales
- Programa GPU
- Texturas 2D

- Compositor
- Fuente

Todos estos tipos de recursos tienen sus tipos particulares de *ResourceManager* (por ejemplo: *MaterialManager*, *FontManager*, entre otros), pero a no ser que se esté escribiendo un nuevo plug-ins o adicionando nuevos tipos de recursos para el sistema de administración de recursos de Ogre, no se va a tener que tratar con el *ResourceManager* para nada.

El *ResourceGroupManager* es el responsable de encontrar los recursos cuando son buscados por su nombre. Él no necesita realizar las tareas de gestión de memoria actual, requeridas para el administrador de recursos (tal como descarga de viejos recursos para dar cabida a nuevos cuando sea necesario); esto es manejado por la clase *ResourceManager*. El *ResourceGroupManager* también permite cargar y descargar grupos de recursos por su nombre.

Por defecto, Ogre espera que sus recursos existan como un fichero. Por lo cual centenares de tipos de recursos pueden ser gestionados, actualmente solo los tipos de recursos como mallas y fuentes, tienen un intérprete de recursos implementado. Pero si se necesita crear un cargador de recursos de forma manual para un determinado tipo de recurso el *framework* está disponible para hacerlo.

Manipulador de escena

En Ogre el grafo de escena es parte del gran concepto conocido como manipulador de escena. El manejador de escena es, como su nombre lo indica, el encargado de la creación, destrucción y manipulación de todos los objetos de la escena. Todas las implementaciones del grafo de escena son derivadas de la clase manipulador de escena. Se tendrá que interactuar con el manipulador de escena activo, dado que Ogre soporta múltiples manipuladores de escena simultáneamente. Por lo cual la mayoría de las aplicaciones crearán y usarán individualmente un manipulador de escena al mismo tiempo.

Nodo de escena

El nodo de escena es el elemento estructural en el grafo de escena de Ogre, en ellos está todo lo que se mueve alrededor de la escena. También puede organizarlos jerárquicamente, lo que facilita el posicionamiento y animación de los objetos en la escena. Los nodos escena pueden existir independientes del grafo de escena, una simple manera de prevenir el *render* de algún contenido de la escena, es separarlo de la jerarquía del grafo de escena, el contenido no se afecta y se puede volver a adjuntar posteriormente.

El contenido es adjuntado a los nodos escena. Casi todo el contenido debe estar en forma de instancia de entidades. Una vez que se tenga una entidad válida se puede adjuntar a un nodo escena. Las entidades son cargadas la mayoría de las veces del fichero, donde existe como archivo binario (.malla). Sin embargo es posible crear manualmente objetos de contenido tales como los planos móviles. Una vez adjuntado el contenido al nodo escena, es el nodo el que se mueve alrededor de la escena y no el contenido.

También es posible adjuntar otros objetos no contenidos al nodo escena. Por ejemplo, puede existir la necesidad de adjuntar una cámara al nodo escena. También se puede adjuntar luces si se desea a un nodo escena [4].

1.4 Determinismo e Indeterminismo

En su concepto más general podemos definir al determinismo como:

“Doctrina filosófica que preconiza la tesis que entre todos los acontecimientos hay una relación ineludible de causa a efecto, especialmente en cuanto a herencia y ambiente como condicionantes de las posibilidades del ser humano, el que de ninguna manera puede sustraerse a esta relación preestablecida. Se le opone la doctrina del libre albedrío [6] .”

A diferencia del determinismo, el indeterminismo niega que todo lo que sucede tenga una causa. Según el indeterminismo, nada sucede "necesariamente", o algunos acontecimientos por lo menos tienen lugar de modo "no necesario". Así, el indeterminismo se opone en todos los casos al determinismo [5].

1.4.1 Determinismo e Indeterminismo Científico

Durante el siglo XIX y XX la formulación más famosa del determinismo es la de Pierre Simon Laplace. Laplace en el "Prefacio" a su *Théorie analityque des probabilités* escribió:

"Una inteligencia que conociera en un instante dado todas las fuerzas que animan a la naturaleza y la situación respectiva de los seres que la componen, si por otra parte fuese lo suficientemente capaz como para someter todos esos datos al análisis, en una misma fórmula llegaría a englobar los movimientos de los cuerpos más grandes del universo, así como los del átomo más ligero: nada sería incierto para ella, y el porvenir y el pasado estarían presentes ante sus ojos. El espíritu humano ofrece, en la perfección que ha sabido dar a la astronomía, un débil esbozo de dicha inteligencia [7]." Este determinismo laplaciano afirma que el estado del universo en un momento dado, futuro o pasado, está completamente determinado si su estado, su situación, es dado en algún momento, por ejemplo, el momento presente.

El determinismo científico puede definirse como: la doctrina que dice que el estado de cualquier sistema físico cerrado en cualquier instante futuro dado puede ser predicho, incluso desde dentro del sistema, con cualquiera que sea el grado estipulado de precisión, mediante la deducción de la predicción a partir de teorías, en conjunción con condiciones iniciales cuyo grado de precisión requerido puede calcularse siempre de acuerdo con el principio de poder dar razón si la tarea de predicción es dada .

El indeterminismo científico es defendido en la ciencia moderna por la mecánica cuántica y aunque existen diversas interpretaciones de la misma, la mayoría de ellas aceptan que existe uno o varios factores aleatorios intrínsecos en la teoría, por los cuales no existiría determinismo como en el caso de la mecánica clásica. En especial, en la reducción o colapso de la función

de onda relacionado con el problema de la medida, se cree que podría ser un proceso donde interviene el azar de manera insoslayable.

Si el determinismo clásico está dado con la definición de estado mecánico, lo cual no deja de ser una convención, el indeterminismo científico consistirá en negar la posibilidad de tal definición, por no contener en sí todos los parámetros que realmente toman parte en un fenómeno. Pero además, negará que tal definición sea posible tal como se hace en la Ciencia clásica, porque tampoco se podrán precisar simultáneamente las llamadas «variables de estado»[8]

A pesar de que diferentes teorías científicas se han combinado para demostrar que el determinismo estaba equivocado, en el universo simplificado de un ARV, el determinismo de Laplace realmente funciona, si se logra grabar todo lo que afecte la dirección del universo en la ARV, se puede reproducir la grabación y recrear todo lo sucedido.

1.4.2 Factores que Crean Indeterminismo en Programas de Ordenador

Los programas de ordenador por naturaleza son totalmente deterministas y previsibles, sin embargo, con frecuencia se pueden encontrar errores que son difíciles de reproducir, por tal motivo se hace difícil de solucionar. Si las computadoras son deterministas, ¿cómo pueden ser los errores difíciles de reproducir? En ocasiones, el culpable es el *hardware* o sistema operativo. Sin embargo, estos raros errores son causados con más frecuencia por la combinación de entradas del usuario. En la siguiente sección se describen cuales pueden ser las causas de que ocurran estos errores en los programas de ordenador.

1.4.2.1 Entradas

Una entrada es un dato que se le pasa a una aplicación en tiempo de ejecución de una fuente externa, que se utiliza de alguna manera para modificar los datos internos. Un ejemplo evidente son las entradas de datos de un usuario. Los resultados son los datos que son generados por

la aplicación y pasan al exterior, pero que no se utilizan internamente por el sistema. Un ejemplo de salida podría ser el vértice de datos pasado a la Unidad de Procesamiento Gráfico (GPU) para ser renderizado. A los restantes datos persistentes internos se les conoce como "Estados del sistema". Ejemplos de los datos de estado de un SRV pueden ser la velocidad de un automóvil, o la posición del carácter en el mundo. Habiendo definido estos términos, se pueden identificar en un motor de juego. La Figura 1.1 representa una simple entrada / salida / estado de acuerdo a un típico motor de juego.

Figura 1: No se encuentran elementos de tabla de ilustraciones[9].

Una vez que haya restaurado el estado inicial del sistema, debe asegurarse de que puede grabar y reproducir todos los datos que puedan afectar al mismo. Si el lazo de actualización de su sistema, está llamando directamente a las funciones del Sistema Operativo para obtener las entradas del usuario. En lugar de ello, todas las entradas tienen que ir a través de un sistema de entradas. Este sistema puede registrar el estado de todos los dispositivos de entrada, al comienzo de cada fotograma, y entregar esta información a lo solicitado por el bucle de actualización del sistema [9].

Tiempo

El tiempo es otra "entrada" que un SRV puede utilizar. La mayoría de los SRV se ejecutan en tiempo real. Puede que ciertos eventos ocurran en un momento determinado, y es importante que estos tiempos se midan en el tiempo del sistema, no en tiempo real. La animación producida por el motor de juego está hecha a escala para las lecturas del temporizador del sistema de *hardware*. Normalmente, se dicta una cantidad diferente de tiempo para hacer cada fotograma, y así se mantiene una animación fluida e independiente de la velocidad de los fotogramas.

Una forma típica de implementar la frecuencia de animación es actualizar el estado del sistema una vez por cada fotograma renderizado. Al comienzo de cada actualización del sistema el temporizador se lee, y el tiempo que ha transcurrido desde la última actualización se calcula. Este " tiempo del último fotograma " se utiliza para calcular el próximo estado del sistema y, a continuación, el fotograma siguiente se renderiza, y así sucesivamente. Por ejemplo, si un coche se mueve con velocidad *V* en el mundo de un juego, y su posición en el último fotograma se dictó 'P1', entonces su posición para el próximo fotograma, P2, se puede calcular utilizando el sencillo método de integración:

$$P2 = P1 + V * \text{tiempo del último fotograma}$$

En la ecuación anterior se puede ver que el resultado de cada estado de transición está determinado por el último fotograma de tiempo medido. La secuencia de los estados del sistema que pasan por el motor depende directamente de las lecturas de tiempo del temporizador del sistema. A pesar de que la animación es independiente de la frecuencia de

renderizado, esta dependencia socavará la reproducibilidad del sistema, porque no se puede confiar en obtener la misma frecuencia de los fotogramas durante la reproducción.

Por ejemplo, imagine que se hace renderear dos fotogramas a un motor gráfico, y que el tiempo para la actualización de cada fotograma es de 30 milisegundos. Esto dará lugar a dos cambios de estado, cada uno representa una actualización de 30ms. Ahora imagine que se quiere reproducir esta secuencia, como se ve desde otro ángulo en el mundo. Desde este nuevo punto de vista, se puede ver un menor número de objetos del mundo y, en consecuencia, hace que el motor tenga una mayor velocidad entre fotogramas. Tal vez toma solo 20 ms el *render* de un fotograma. En lugar de generar 2 fotogramas de 30ms, el motor independiente de la velocidad del fotograma ahora genera 3 fotogramas de 20 ms. El resultado de esto es que la posición del coche es ahora un poco diferente. A pesar de que ha pasado exactamente la misma cantidad de tiempo, los errores en la integración de la velocidad, significan que su posición es ligeramente diferente. En la repetición, el coche no se pasa por el mismo conjunto de estados como lo hizo la secuencia original. Además se tiene un problema reproduciendo las entradas de un usuario. Si las entradas se leen una vez por fotograma, entonces no se pueden aplicar exactamente a la misma vez en el *replay* de la forma en que se hizo en el sistema original.

Tratar con estas lecturas del tiempo del sistema es crítico para la reproducibilidad. Incluso cuando se ejecuta una repetición en el mismo equipo que se haya generado, no hay garantía de que las mismas lecturas del temporizador sean generadas nuevamente. Incluso si se hace un pequeño cambio de punto de vista dará como resultado diferente lecturas, y de esa forma, una secuencia diferente de estados del sistema. Este problema se agrava cuando se ejecuta un sistema basado en PC, porque el sistema operativo puede ejecutar otros procesos externos, o el usuario puede, quizás, tomar la decisión de actualizar su tarjeta gráfica para conseguir una mayor velocidad de fotograma [9].

Red en los Juegos

En el caso de un sistema multi-jugador, Si se quiere lograr la reproducción, se necesita registrar la entrada de los datos de la red junto con el flujo de entrada del usuario. Esto

permitirá reproducir el juego, incluso sin una conexión de red. La aplicación de un exitoso sistema en red puede ser muy problemático. A fin de mantener la respuesta, es generalmente necesario ejecutar una versión local del mundo del juego en cada máquina, y transmitir los datos generados a nivel local a los demás participantes. La latencia inherente a cualquier sistema de transmisión, significa que la sincronización es inevitable. Eventualmente esta sincronización se traducirá en un evento que ocurre en una máquina y no se produce en la otra, por lo que es necesario aplicar algún sistema de re-sincronización [9]

1.4.2.2 Estado Inicial

Se debe asegurar que el sistema comience en un estado conocido, si ejecuta un nuevo sistema o se carga alguno guardado, esto normalmente ocurre automáticamente. Sin embargo, cada vez que se recopilen o cambien datos se está modificando ligeramente el estado inicial. Afortunadamente, muchos cambios de código y datos no afectan la forma en que el sistema se desarrollará. Por ejemplo, si se modifica el tamaño de una textura, entonces la velocidad de fotograma puede ser diferente, pero el comportamiento no, siempre y cuando el sistema sea predecible. Si se modifica el tamaño de esa textura provoca que todos los otros bloques de memoria sean asignados a diferentes lugares, entonces esto tampoco debería tener efecto, siempre y cuando el código no tenga ningún error para sobrescribir memoria.

Un ejemplo de un código o de datos que con su cambio podrían afectar la forma de cómo se comporta el sistema, en este caso de un juego, sería modificar la posición inicial de una criatura, de una pared, o ajustar ligeramente la probabilidad de un determinado evento. Pequeños cambios nunca podría hacer una gran diferencia, pero destruiría la garantía de la predictibilidad [10].

1.4.2.3 Puntos Flotantes

En algunos casos, el uso de las matemáticas de coma flotante también puede causar problemas con la reproducibilidad, son un área en la que sus resultados pueden

inesperadamente variar. El problema surge cuando se quiere reproducir una secuencia en una plataforma diferente a en la que se ha generado, o por lo menos, una plataforma de *hardware* con una unidad diferente de punto flotante. En una plataforma de *hardware*, como la consola, no se presentan problemas. Sin embargo, si se está desarrollando un juego para ser ejecutado en una PC, se debe ser consciente de que una PC no es una plataforma estándar.

Distintos sistemas de PC utilizan diferentes unidades centrales de procesamiento (CPUs), y distintos CPUs tienen diferentes Unidades de Punto Flotante (FPUs). Puede parecer sorprendente, pero distintos FPUs puede producir resultados ligeramente diferentes para algunos cálculos, a pesar de que todas las unidades son compatibles con el Institute of Electrical and Electronics Engineers (IEEE). Esto puede atribuirse a variadas representaciones internas dentro de la FPU. Además, desarrolla diversos programas que pueden mostrar un comportamiento diferente en el mismo *hardware*, debido a los cambios en el almacenamiento de valores de punto flotante. Esto significa, por ejemplo, que una secuencia reproducida a través de una construcción "depurada" de un juego, podrá exhibir un comportamiento que no es igual a cuando se juegue a través de una construcción de "entrega". En cualquier caso, esto deja un problema si se quiere reproducir una secuencia de juego en una PC que no sea en la que se registró [9].

1.4.2.4 Otras Fuentes de Datos Externos. Números Aleatorios

"Datos externos" esencialmente significa "cualquier otra fuente de datos externos generados en tiempo de ejecución", y puede incluir *hardware* o fuentes de sistemas operativos, o bibliotecas de *software* utilizadas por el sistema. Estos son específicos para cada aplicación y por tanto deben considerarse caso por caso. No obstante, lo importante es que no perjudique la reproducibilidad. Cualquier fuente de datos externos utilizados por el sistema tiene que ser reproducible. También es importante que pueda ser restaurado por el sistema cuando es cargado [9].

Una fuente generadora de números aleatorios que se utiliza en muchos sistemas es la C / C++ generador de números aleatorios de la biblioteca estándar. Este es reproducible, siempre

genera la misma secuencia con la misma semilla. Sin embargo, hay un problema salvando y cargando el estado de un sistema.

Cuando una aplicación se carga, es claramente necesario restablecer el generador de números aleatorios, para que genere la misma secuencia a partir de ese momento, como lo hizo después de que fueron salvados los datos de la aplicación. Una solución sería la tentación de volver a las semillas del generador, cada vez que se produzca una salva del sistema y guardar la semilla como parte de los estados guardados.

Como se mencionó anteriormente, los números aleatorios pueden ser utilizados en un sistema determinista, pero hay algunas salvedades. La razón de que los números aleatorios se pueden emplear es que las funciones que los generan no son realmente aleatoria, se implementa mediante un simple algoritmo, normalmente es un método lineal congruente que pasa muchas de las pruebas de detección de números aleatorios. Esto se llama un pseudo-generador de números aleatorios. Al inicializar esta función con una semilla, se obtendrá resultados consistentes. Si la aleatoriedad es diferente en cada momento del sistema es importante entonces, guardar la semilla que se utilizó para generar la secuencia aleatoria de modo que se pueda reutilizar, si se necesita volver a reproducir el sistema.

Uno de los problemas con estas funciones es que producen un único flujo de números aleatorios, por ejemplo, si el código de renderizado y la actualización del código del sistema es utilizando la función generadora de números aleatorios de la biblioteca estándar (rand) y si el número de fotogramas renderizados por el sistema varían, entonces la actualización del estado del generador rápidamente se convertirá en indeterminado.

Continuando con el ejemplo de *rand* es que su comportamiento no es portátil. Es decir, el comportamiento no está garantizado a ser idénticos en todas las plataformas, y es poco probable que lo sea. Otro problema con *rand* llega si se guarda un sistema, se continúa ejecutando, y luego se quiere volver a cargar lo guardado y reproducir las futuras entradas. Para hacer este trabajo predecible, se tiene que poner el generador de números aleatorios de vuelta al estado en que estaba cuando fue guardado el sistema. El problema es que no hay forma de hacer esto. Las normas de C y C + + no dicen nada acerca de la relación entre los

números que salen de *rand ()* y el número que se necesitan para enviar a *srand ()* y ponerlo de nuevo a ese estado [10].

1.5 Las Cámaras en un *Replay*

Quizás, la parte más importante en un *replay* es la representación visual, debido a que el usuario no está interactuando con el sistema durante el mismo, y es muy importante que él o ella se mantengan interesados. Si la atención del usuario se pierde por la frustración o por el aburrimiento, puede que no utilice el sistema nuevamente. La óptima utilización de las cámaras en el *replay* juega un papel relevante en evitar estos dos aspectos fundamentales que crean descontento en los resultados generados.

1.5.1 Causas Fundamentales de Descontento en los Resultados Generados en un *Replay*

1.5.1.1 Frustración

La primera gran cuestión es la frustración. Muchas cosas pueden causar que el usuario se sienta frustrado con el *replay*, una de la cuales es estar desprovistos de exactitud. Si los actores en el *replay* se comportan en una manera obviamente diferente de cómo lo hicieron durante el juego, el jugador pierde la confianza en el solución. Esto puede remediarse con el uso de un sistema de grabación que se ajuste a sus necesidades y que garantice la validez de los datos.

Otra razón para la frustración es el pobre funcionamiento durante el *replay*, causado por el inadecuado ancho de banda por donde fluyen los datos o por la carencia de una adecuada memoria del sistema. El adecuado diseño y optimización del código relacionado puede remediar estas cuestiones. Aunque el usuario no interactúe con el sistema durante el *replay*, es importante que sienta el control del mismo en todo momento. Si el *replay* se inicia y solo se tiene la opción de detenerlo, esto da una mala experiencia. El sistema debe permitir al usuario un mayor control del *replay*, si así se desea.

Algunos posibles controles a incluir para el control de un *replay*:

- Reiniciado.
- Pausa/Volver.
- Incrementar /Disminuir la velocidad del *replay*.
- Saltar Hacia delante/Hacia atrás por un intervalo fijo.
- Cambiar manualmente a diferentes cámaras.
- Alternar la exhibición principal de la pantalla.
- Abandonar el *replay*.

1.5.2.2 Aburrimiento

Otra cuestión fundamental con un *replay* es el aburrimiento. Una forma garantizada de evitar el aburrimiento, es que el *replay* tenga un ritmo rápido e interesante. Pensando como si estuviéramos haciendo un video musical. Muchos de los trucos cinematográficos pueden ser adaptados en un *replay* para prevenir el aburrimiento. Sin embargo, es diferente a los estudios cinematográficos; no se tiene el lujo de grabar montones de contenidos y editarlos con bastante tiempo en un estudio; se tiene que hacer todo en tiempo real.

Aquí es donde una buena regla basada en inteligencia artificial puede ayudar. El problema ha sido bien estudiado desde el punto de vista cinematográfico, y un conjunto de reglas básicas para el trabajo de cámara son establecidas. Además porque se tiene una buena idea acerca de la escena, actor, y como se comportan, el problema es muy bien solucionado con las reglas basadas en inteligencia artificial.

1.5.3 Ángulos y Estilos de la Cámara

Como se mencionó antes, el mayor peligro que afronta el *replay* de un SRV es el aburrimiento.

Haciendo un *replay* con un ritmo rápido e interesante garantizará que los usuarios esperen a verlo. Variando la experiencia visual, es una vía para mantener el interés del usuario. Esto se

puede lograr con un enorme arsenal de interesantes ángulos y estilos de cámara, mezclándolas usando reglas especializadas.

A continuación se muestra una serie de tipos de cámaras y sus variaciones, así como reglas para usarlas.

Cámara introductoria:

Esta cámara introduce la escena y los actores, comienza a una distancia y un andar sin rumbo, y rápidamente se aproxima al sujeto, poniendo en escena como llega al lugar inicial. Después aproximándose al sujeto haciendo un círculo alrededor de él, mostrando los detalles del sujeto antes de tomar la posición en la que estará la otra cámara. Esta cámara no deberá estar activada por más de 3-5 segundos. Ella deberá completar su rol y terminar para evitar el aburrimiento. Este tipo de cámara está limitada solo al comienzo del *replay* cuando el sujeto está estacionario, nunca se debe usar después.

Cámara en Primera Persona:

Esta cámara presenta la escena desde el punto de vista del sujeto, es muy usada en los videojuegos de carreras donde da la sensación de que efectivamente el carro esta junto al chofer. Y también se utiliza con frecuencia en los videojuegos de deportes, donde el sujeto está interactuando con otros actores en la escena. Las variaciones de esta cámara en la escena actual presentan una posición sobre o muy cerca del avatar del sujeto; por ejemplo el frente del timón de un carro, o el casco de un jugador de pelota. Es muy importante tener un punto de referencia en el sujeto, visible en esta vista de cámara, por otro lado esta vista puede ser muy desconcertante. Esta cámara se puede usar con frecuencia pero debería ser interrumpida dentro de 2 – 4 segundos para evitar los mareos.

Cámara de Seguimiento:

Está referida a una cámara en tercera persona. Típicamente muestra las partes diferentes partes de la escena, la parte trasera del sujeto, simulando mirar por encima del hombro del mismo. Las variaciones en esta cámara pueden presentar al sujeto desde varias distancias,

alturas y desde varios campos de visión. Las variaciones pueden ser lo suficientemente diferentes como para terminar en otra variación no deseada, que puede causar una anomalía.

Otra variación interesante de la cámara es creada cuando no está siempre en una distancia fija y el sujeto da un acercamiento inesperado. Esto permite al jugador alejarse con un atraso de reacción de la cámara. Sin embargo los saltos pueden causar mucha molestia cuando las tazas de refrescamiento no son constantes. Además, debe usarse con cuidado para evitar que sea brusca la oscilación durante el *replay*.

Se puede usar en más de un 50% del tiempo de un *replay*, lanzando otras vistas de cámaras interesantes entrelazándolas entre sí. La secuencia puede estar entre 3-9 segundos. También puede ser usada como una sesión de calma después de una sesión repleta de cambios continuos de vistas de cámara.

Cámara Retrovisor:

Es la vista inversa de la cámara de seguimiento. La cámara va delante del sujeto a una cierta distancia y correspondiéndose con su velocidad y dirección de movimiento, mostrando efectivamente al sujeto y a la escena que está detrás de él. Las variaciones pueden ser creadas para permitir que la cámara se mueva a diferentes velocidades del sujeto, de esta manera permite un acercamiento y alejamiento del sujeto. Además este agradable efecto cinematográfico es creado para mover una cámara con un rango fijo de duración en la parte de una toma. Sin embargo si el sujeto cambia de dirección rápidamente, esta vista de cámara puede causar mareo, debería ser empleada en periodos de tiempos cortos (2-4 segundos), o ser usada cuando el jugador se esté moviendo en una dirección constante.

Cámara Lateral:

Rastrea al sujeto mientras está a su costado imitando su velocidad y dirección. Permite mover la cámara a diferente velocidad que la del sujeto, creando variaciones que resulten interesantes. Esto pudiera simular una cámara adelantándose a un jugador, o un jugador adelantándose a la cámara, cualquiera de los dos son buenos efecto cinematográfico. Este tipo

de cámara provee al jugador de una interesante vista que no es visible durante el juego. Se debe limitar esta cámara de 3 a 5 segundos de tiempo.

Cámara Baja o Panorámica:

Este tipo de cámara está en localizaciones fijas, dando un panorama de cómo el sujeto pasa por allí, simulando el efecto de un espectador. Debe ser forzada a cambiar cuando el sujeto logra alejarse, para evitar ver pequeñas cantidades de movimiento en la distancia. Las variaciones de esta cámara pudieran ser el movimiento hacia abajo en varias direcciones pasando por el sujeto. Esta es una típica técnica utilizada en los desfiles de televisión. Este tipo de cámara provee una interacción escena-sujeto a diferentes localizaciones en la escena, haciendo lo suficientemente interesante como para ser usada con frecuencia. Debe ser interrumpido una vez que el sujeto no esté visible o esté lejos.

Cámara Baja y de Seguimiento:

Esta es una variante de la cámara baja estática, que permite al sujeto obtener una cierta distancia por delante de él, y entonces se recoge y se convierte en una cámara de seguimiento. Esto provee de una buena transición cambiando a estático o una entrada lenta a una escena cargadas de secuencia de acciones.

Cámara de Búsqueda y Seguimiento:

Es similar a la cámara introductoria, excepto que el sujeto se está moviendo. Comienza a una distancia de giro, aumentando la velocidad hasta estar en correspondencia con el sujeto. Esto es a una distancia fija, siendo intercalada con la cámara de seguimiento, esta cámara puede desorientarse. Para evitar esto, el jugador debe mantenerse a la vista. La distancia y la velocidad pueden ser ajustadas de modo que la secuencia no durará más de 2 a 4 segundos.

Cámara “Matrix”:

Esencialmente fue popularizada por la película “La Matrix”. La cámara pausa en todas o en algunas partes de las acciones, o demorándolo haciendo un movimiento circular alrededor del sujeto. Da un efecto dramático y pudiera ser usada cuando el sujeto hace algo espectacular.

Debe usarse con moderación, solo en pocas ocasiones durante la duración del *replay* completo.

Cámara Circular:

Realiza movimientos circulares lentos desde la perspectiva del sujeto. Esto es ideal para un intervalo de poca actividad en el *replay*. Puede ser usada como la transición de cámara que va desde un lado del jugador a otro, de izquierda a derecha, o de adelante hacia detrás.

Cámara de Imagen Fija:

Ha sido muy popularizada por las películas y es otra variación de la cámara estática panorámica. Es como el paso del sujeto, la cámara pausa la acción en intervalos regulares simulando una secuencia de toma de fotos. Efectos visuales tales como: fotos en blanco y negro o el sonido de la toma de una foto pueden enriquecer los efectos de esta cámara.

Cámara de Colocación:

Esta cámara aplica reglas para los *replay* que son basados en una escena fija, donde los actores se ven obligados a estar dentro de ciertos límites. Las cámaras pueden ser estratégicamente emplazadas en la parte privilegiada de la acción, como pudieran ser la meta de una carrera, la portería de un juego de futbol o en los límites de un terreno. Las cámaras pudieran ser puestas ocasionalmente donde un autor ve el sujeto. Dando la sensación de perspectiva a la escena que tributa a un agradable toque cinematográfico.

1.5.4 Cortes de Cámara para SRV, especialmente en Videojuegos

- Una Secuencia de cámara no deberá ser menor de 2 segundos.
- Una Secuencia de cámara no deberá ser mayor de 9 segundos.
- El cambio de cámara nunca debería cortar en el lado opuesto de un jugador, porque es muy desconcertante.
- El sujeto deberá ser siempre visible en una secuencia o por lo menos visualizar la anticipación de la aparición de este.

- El tiempo de anticipación no deberá exceder los 2 segundos.
- La transición de una cámara a otra deberá ser imperceptible, instantáneo, a través de una adecuada transición se evitan los llamados “saltos” en el *replay*.
- Las cámaras especiales como la Cámara Matrix, Cámara de Imagen Fija, Cámara de Búsqueda y seguimiento deben ser usadas con moderación.
- Las cámaras especiales nunca deben ser usadas una a continuación de otra [11].

1.6 Enfoques para Implementar un *Replay*

A pesar del desarrollo existente actualmente en el mundo de los SRV, poca es la bibliografía existente sobre las diferentes maneras de construir un sistema reproducible. Dos enfoques principales son los posibles a utilizar para lograr la misma en un SRV, el conocido como: “salvar estado” y el “determinista”.

1.6.1 Salvar Estados

Este enfoque se basa en acceder a toda la información existente en la escena en intervalos de tiempo pequeños. Se debe salvar a una frecuencia fija (pocos segundos) el estado de todos los objetos que se encuentren en la escena, fotograma a fotograma. La secuencia es reconstruida enviando la información de vuelta al sistema e interpolándola en caso necesario.

Desventajas:

El principal problema de este enfoque es la cantidad de requisitos de memoria que se necesitan para almacenar todos los datos necesarios para la reproducción del sistema.

Como ejemplo tenemos un videojuego de carreras de autos para obtener algunos números concretos: si el promedio de velocidad de un fotograma es 60 Hz y la duración media de una carrera es de 300 segundos, se tendrá que generar 180,000 fotogramas para reproducir el videojuego. Para algunos géneros de videojuego, la duración podría ser incluso mucho más tiempo.

Dentro de este mismo videojuego, otro aspecto a analizar sería la cantidad de memoria que se necesita para salvar el estado (lo que llamamos una "instantánea") del videojuego en cualquier momento dado. Por cada vehículo se tienen que salvar sus propiedades físicas, al igual que las matrices de inercia, el vector velocidad, las ruedas de agarre, los parámetros de inteligencia artificial (IA) y así sucesivamente. Almacenar el estado de un vehículo requiere algo así como 1K. En este videojuego, una carrera cuenta con seis vehículos. Almacenar el estado del circuito requiere de otros 4Kb. Aproximadamente se requiere de 10K para almacenar el estado de la carrera. En resumen se requieren alrededor de 200MB para almacenar 180,000 fotogramas[12].

1.6.2 Determinista

Este enfoque se basa en salvar solo un número limitado de variables y en el comportamiento de su código. Solamente se necesita salvar el estado inicial del sistema y los diferentes estados del controlador de la escena, además, el sistema debe tener un código con comportamiento determinista. Teniendo almacenado este número reducido de datos y un comportamiento determinista en el código se puede logar la misma secuencia de reproducción que la creada originalmente por el sistema

Desventajas:

El principal inconveniente para el enfoque determinista es que si alguna parte del videojuego no es determinista, el resultado de la repetición puede diferir de lo ocurrido originalmente, varios son los aspectos que pueden crear indeterminación en el sistema y con ello provocar impredecibles cambios en la repetición, lo cual provocaría que fuera totalmente diferente a lo ocurrido originalmente, por eso debe ser una tarea priorizada el crear un sistema con comportamiento determinista, si se desea implementar en él la funcionalidad de reproducción, algo que no es nada sencillo si no se tienen bien delimitadas las partes y factores deterministas y no deterministas dentro del sistema [12].

Conclusiones

Al concluir este capítulo se cuenta con un estudio del estado del arte y los enfoques que actualmente existen para dar solución a la construcción de un *replay* ofreciéndose las principales ventajas y desventajas de cada uno.

Se destacaron aspectos importantes como el determinismo e indeterminismo y las estructuras de datos empleadas por diferentes motores gráficos para el manejo de los objetos en la escena, aspectos que son necesarios para entender el proceso de obtención de un *replay*.

Capítulo 2: Soluciones Técnicas

Introducción:

Habiendo concluido el estudio del estado del arte sobre los diferentes enfoques existentes para la creación de un *replay*, en el presente capítulo se presentará una solución para dar cumplimiento al objetivo planteado en este trabajo, la cual estará compuesta por uno de los enfoques estudiados en el capítulo anterior.

2.1 Enfoques

Solamente dos enfoques, los cuales fueron explicados en el capítulo anterior, son los empleados para la obtención de un *replay*, el primero plantea que salvando sucesivamente fotograma a fotograma toda la información existente en la escena y devolviéndola de nuevo al sistema es reconstruida una secuencia igual a la ocurrida originalmente, el segundo consiste en salvar solamente un número limitado de variables con las cuales si el comportamiento del código del sistema es determinista se obtiene la misma secuencia de reproducción que la creada originalmente por el sistema. El primer método tiene como principal inconveniente lo costoso que puede ser en cuestión de memoria por la cantidad de información que se necesita almacenar y el segundo la necesidad de tener un código con comportamiento determinista para poder con la información almacenada reproducir el *replay* del sistema. Finalmente este trabajo propone el enfoque determinista, es evidente que la utilización de este conlleva a un esfuerzo para crear un sistema con comportamiento determinista del código, pero cabe resaltar que este enfoque debe ser utilizado siempre que se desee implementar un sistema de repetición en una aplicación de realidad virtual, los resultados de los cálculos realizados en la sección 1.6.1, del capítulo anterior demuestran los costos de utilizar el enfoque “Salvar Estados”.

2.2 Determinista

El enfoque determinista cuenta con dos fases fundamentales, la primera se encarga de grabar un grupo de valores, los cuales definen el comportamiento del sistema y la segunda fase se encarga de restaurar al sistema los valores que fueron grabados con anterioridad, el cual al tener un comportamiento de código determinista reproducirá exactamente la misma escena que se generó originalmente.

2.2.1 Comportamiento Determinista del Código

Como se mencionó en la sección anterior un factor indispensable para implementar un *replay* utilizando el enfoque determinista es lograr que el sistema al cual se le quiera implementar esta funcionalidad tenga un comportamiento de código determinista. Existen un gran número de factores que pueden crear indeterminismo dentro de un SRV (ver sección 1.4.3) y es una tarea muy difícil detectar en un sistema ya terminado cuál de estos factores crea este efecto en algún momento, por lo tanto, lo más recomendable si se desea implementar un *replay* a un sistema, es diseñarlo desde un principio para que tenga un comportamiento de código determinista, para lo cual no es necesario que todas las partes del sistema lo sean, si no, solo las que lo necesiten, esto se debe a que no todo lo que ocurra dentro del sistema va a generar algún valor de vuelta que pueda incurrir en algún cambio dentro del mismo, por lo tanto no es necesario que tenga comportamiento determinista dado que no tendrá implicación alguna en la ejecución del *replay*, se debe identificar en el sistema la frontera entre las partes que necesitan serlo y las que no. En la siguiente figura se muestra el diseño de un sistema que contiene la funcionalidad de *replay* identificando las partes deterministas del mismo.

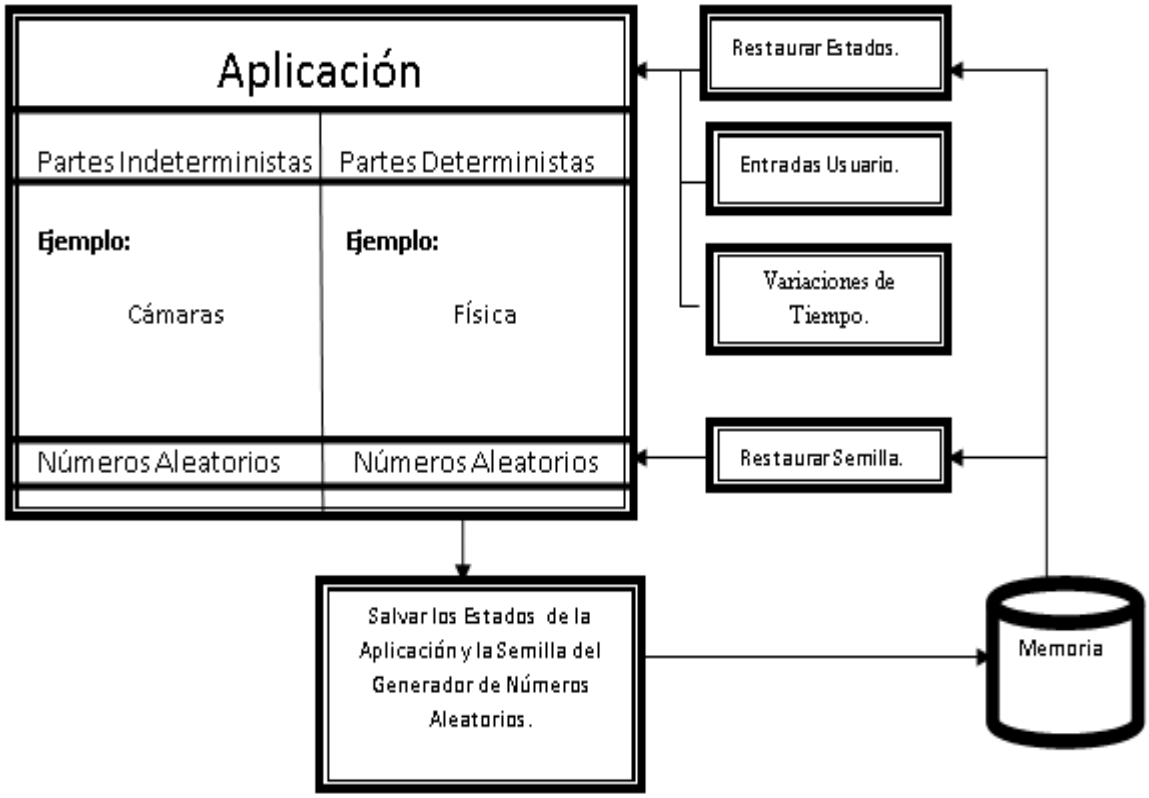

Figura 2: Diseño de sistema, identificando las partes deterministas[12]**.**

2.2.2 Procedimiento General de un *Replay* Utilizando el Enfoque Determinista

El procedimiento general para lograr el *replay* en un SRV utilizando el enfoque determinista consta de los siguientes pasos:

El primer paso es realizar la grabación de la escena que se va a reproducir, para ello en primera instancia se almacenará en memoria el estado (orientación y posición) de todos los objetos presentes en la misma en el momento que comience la grabación, así como las

entradas de los usuarios (interrupciones de teclado o de ratón), el instante de tiempo en que se inicia la grabación y la duración de las interrupciones, mientras se continúe grabando solo se guardarán las nuevas entradas de los usuarios (interrupciones de teclado o de ratón) y la variación de tiempo en que ocurran. Al culminar la grabación los datos que fueron almacenados pueden ser salvados si se desea, si este es el caso todo lo que fue almacenado en memoria quedará guardado en un fichero. Por último se reproducirá la escena de acuerdo a los datos que fueron grabados, esto sucederá pasándole al sistema todos estos datos, ya sea directamente de memoria, donde fueron almacenados durante la grabación o del fichero donde fueron salvados, con los cuales primero se reconstruirá la escena y se continuará reproduciendo el *replay* de acuerdo a los datos almacenados.

Durante la reproducción del *replay* se pueden ejecutar un grupo de acciones en tiempo real. En la siguiente figura se muestra el procedimiento.

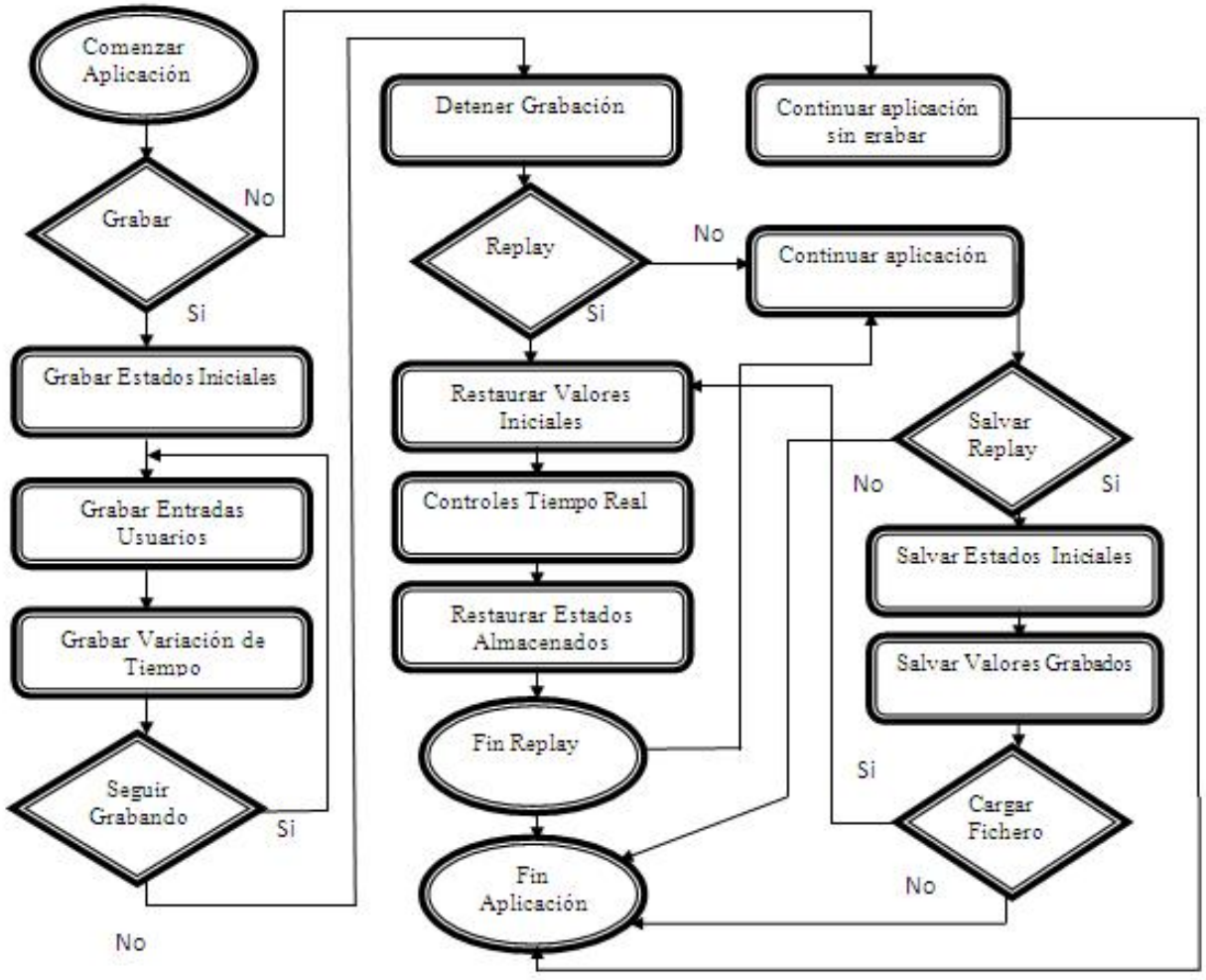

Figura 3: Sistema de *Replay* (Enfoque Determinista)

2.3 Metodologías, Herramientas y Lenguaje Utilizado

2.3.1 Metodología

La metodología aplicada fue el Proceso Unificado de desarrollo de *Software* (RUP) por ser un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de *software* con diversas áreas de aplicación, distintos tipos de organizaciones, diferentes niveles

de aptitud y tamaños de proyecto. Sin embargo, hay tres características fundamentales que lo hacen una metodología robusta y poderosa: es dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. Cuando se caracteriza RUP como dirigido a casos de uso se refiere a que se sigue un hilo, avanzando a través de una serie de flujos de trabajo que parten de los casos de uso. Los mismos se especifican, se diseñan, se implementan y a partir de ellos se obtienen los casos de prueba. El desarrollo de los casos de uso no se hace de forma aislada sino que se desarrollan con la arquitectura del sistema, por lo que tanto la arquitectura del sistema maduran a medida que avanza el sistema. Centrado en la arquitectura se refiere a que se incluyen los aspectos estáticos y dinámicos más significativos del sistema. Además recoge una serie de factores como la plataforma en la cual funciona el *software*, los bloques de construcción reutilizables que se tienen, consideraciones de implementación, sistemas heredados y requisitos no funcionales. Es iterativo e incremental porque el proyecto o el desarrollo de una aplicación se pueden dividir en partes y desarrollarlas de manera iterativa, incrementándose a medida que se integran unas con otras hasta llegar a formar la tarea final. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos en el crecimiento del producto.

2.3.2 Visual Studio 2005

El Visual Studio 2005, es una herramienta poderosa, fuerte y voluminosa que tiene una gran integración de varios lenguajes entre ellos el C++, C#, y Asp.Net. Tiene la posibilidad de implementar aplicaciones para soluciones integrales que aprovechen de manera óptima la ventaja de cada lenguaje. Acelera de manera significativa la producción de *software*. Mejora en un alto grado los resultados finales y optimiza el desempeño. La interfaz es altamente amigable con el usuario permitiendo que el tiempo en implementar una solución o aplicación determinada sea mucho menor. Los ejecutables desarrollados por esta herramienta son generalmente de menor tamaño lo que hace que ocupe un lugar cimero en la producción de *software* al lograr aplicaciones óptimas y de poco volumen.

2.3.3 Lenguaje de Programación C++

El C++ se escogió por ser un lenguaje de programación estandarizado por la norma ISO/IEC 14882:1998. Entre sus principales características está el soporte para la programación orientada a objetos y el soporte de plantillas o programación genérica, además de ser un lenguaje de alto nivel que está considerado como un lenguaje potente al poder trabajar tanto en bajo, como en alto nivel. Posee dos propiedades fundamentales que son difíciles de encontrar en otros lenguajes que son la posibilidad de redefinir los operadores, comúnmente conocido como la sobrecarga de operadores, y la identificación de tipos en tiempo de ejecución. Además presenta una biblioteca estándar muy poderosa, muy usado en el ámbito educacional y profesional, siendo uno de los más utilizado en los gráficos por computadoras.

Conclusiones

En el recién concluido capítulo se propone una solución al problema que se plantea en esta investigación, definiendo entre los enfoques expuestos anteriormente, el más conveniente para aplicar, en aras de lograr un alto nivel de exactitud y eficiencia. Se especifica además la metodología y herramientas de desarrollo de *software* a utilizar.

Capítulo 3: Descripción de la Solución Propuesta

Introducción:

En el presente capítulo se tiene como objetivo guiar el desarrollo del sistema y la compresión del mismo desde la perspectiva de las necesidades del cliente. La tarea fundamental para lograr estos objetivos es la identificación y descripción por parte de clientes y desarrolladores con un total entendimiento de los requisitos que el sistema debe cumplir.

A continuación se presenta el modelo de dominio, requisitos funcionales y no funcionales así como los requerimientos del sistema capturados como casos de uso según establece RUP.

3.1 Modelo del Dominio

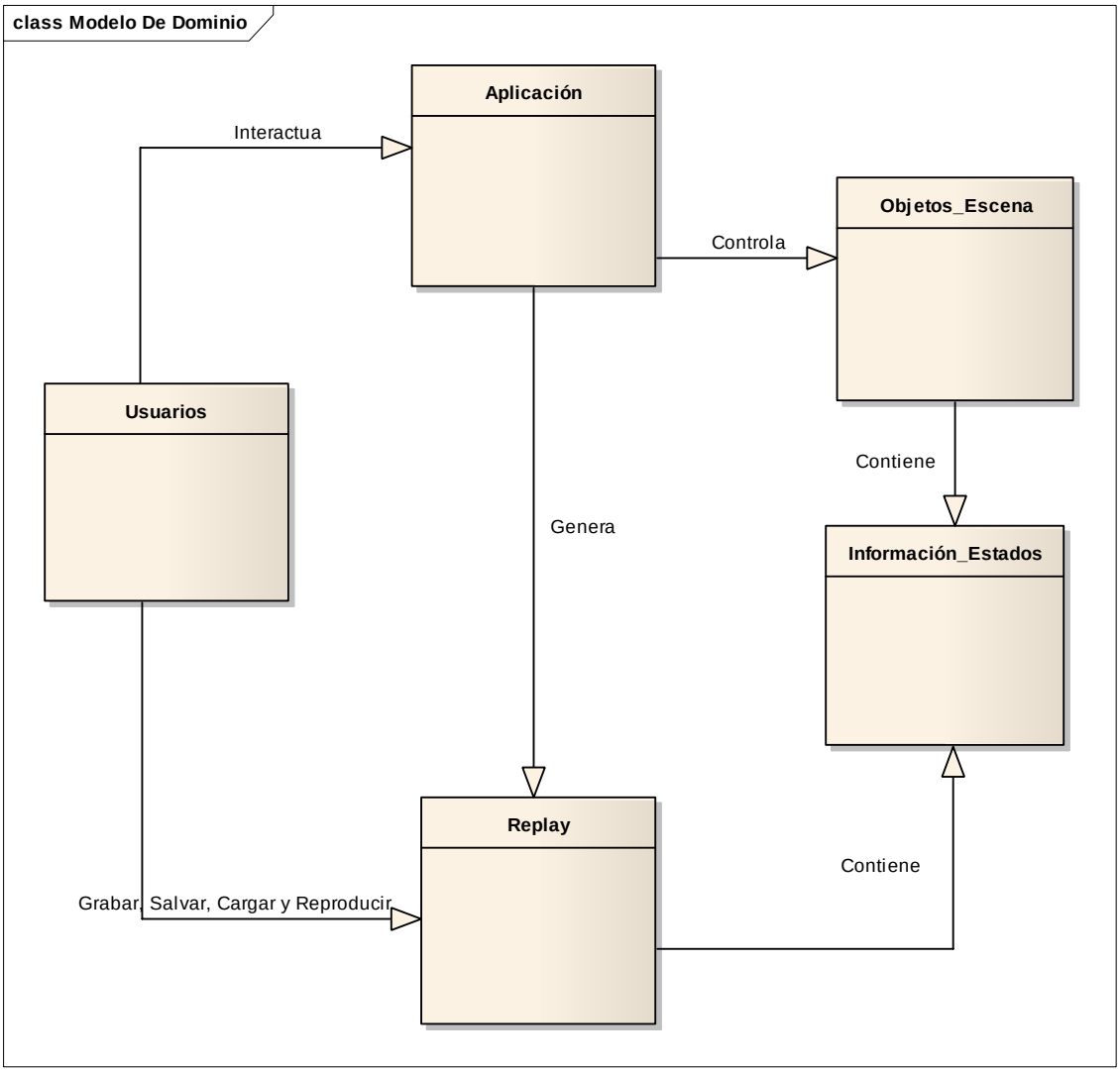

Figura 4: Modelo de Dominio

3.2 Levantamiento de Requisitos

A continuación se expondrán los requisitos funcionales y no funcionales del sistema, los cuales son las condiciones o capacidades que el sistema debe cumplir.

3.2.1 Requisitos Funcionales

R1. Grabar *replay*.

- R1.1 Recorrer grafo de escena.
- R1.2 Almacenar en una lista todos los objetos de la escena.
- R1.3 Grabar estados iniciales de todos los objetos de la escena (posición, orientación e instante de tiempo inicial de la grabación).
- R1.4 Verificar cuando ocurran nuevos eventos.
- R1.5 Grabar estados finales de todos los objetos de la escena (posición, orientación e instante de tiempo final de la grabación).

R2. Guardar *replay*.

- R2.1 Recorrer la lista de la información grabada.
- R2.2 Crear un fichero.
- R2.3 Guardar toda la información almacenada en la lista para el fichero.

R3. Cargar *replay*.

- R3.1 Recorrer la lista con la información que fue grabada en memoria.
- R3.2 Restablecer al sistema toda esta información.

R4. Cargar *replay* desde fichero.

- R4.1 Destruir la información existente en memoria de los objetos de la escena.
- R4.2 Recorrer el fichero seleccionado.

- R4.3 Restablecer a memoria la información almacenada en el fichero.
- R4.4 Recorrer la lista con la información restablecida del fichero.
- R4.5 Restablecer al sistema toda esta información.

R5. Gestionar velocidad de reproducción *replay*.

- R5.1 Aumentar la velocidad con que se reproduce el *replay*.
- R5.2 Disminuir la velocidad con que se reproduce el *replay*.
- R5.3 Mantener a velocidad de grabación la reproducción el *replay*.

R6. Gestionar velocidad de retroceso *replay*.

- R6.1 Aumentar la velocidad de retroceso con que se reproduce el *replay*.
- R6.2 Disminuir la velocidad de retroceso con que se reproduce el *replay*.
- R6.3 Mantener a velocidad de grabación la reproducción del *replay* en retroceso.

R7. Inicio/Fin *replay*.

- R7.1 Saltar al comienzo del *replay*.
- R7.2 Saltar al final del *replay*.

3.2.2 Requisitos No Funcionales

- **Usabilidad:** Los futuros usuarios del sistema serán programadores con conocimientos básicos de programación y de la terminología afín. El producto debe estar concebido para que el usuario piense en qué desea hacer y no en cómo hacerlo.

- **Rendimiento:** Como aplicación de tiempo real, debe tener alta velocidad de procesamiento (alrededor de 60 fps) y cálculo.

- **Soporte:** En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar para Linux.

- **Diseño e implementación**: Debe utilizarse la biblioteca de clase de OGRE la cual tiene definidas las funcionalidades básicas para trabajar con gráficos por ordenador, soporta OpenGL y DirectX. Se harán llamadas a dicha biblioteca desde el lenguaje C/C++. Se regirá por la filosofía de Programación Orientada a Objetos.

  ***Hardware*:** Para su uso óptimo debe emplearse una PC que cuente con tarjetas gráficas de la familia:

  -NVidia: Geforce2 o superior, Geforce 4 o superior.

  -ATI: Radeon 7500 o superior, Radeon 9600 o superior.

- **Legales:** Se utilizarán bibliotecas externas con licencias libres, nada propietario.

3.3 Modelado de Casos de Uso del Sistema

A continuación se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales anteriormente especificados, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema. Además, se seleccionan los casos de usos correspondientes al primer ciclo de desarrollo y se les hacen las especificaciones textuales en formato expandido.

3.3.1 Actor del Sistema

Actores	Justificación
Usuarios	Usuarios que interactúan con funcionalidades que brinda el sistema.

Tabla 1: Actor del Sistema.

3.3.2 Casos de Uso del Sistema

CU1	Grabar *replay*
Actor	Usuarios.
Descripción	Hace una grabación de todo lo sucedido en la escena.
Referencia	R1.1, R1.2, R1.3, R1.4, R1.5 y R1.6.

Tabla 2: CU1 Grabar *replay*.

CU2	Guardar *replay*
Actor	Usuarios.
Descripción	Guardar todos los datos grabados en un fichero.
Referencia	R2.1, R2.2 y R2.3.

Tabla 3: CU2 Guardar *replay*.

CU3	Cargar *replay*
Actor	Usuarios.
Descripción	Reconstruir la secuencia de escenas de acuerdo a los datos grabados.
Referencia	R3.1 y R3.2

Tabla 4: CU3 Cargar *replay*.

CU4	Cargar *replay* desde fichero
Actor	Usuarios.
Descripción	Reconstruir la secuencia de escenas de acuerdo a los datos almacenados en el fichero.
Referencia	R4.1, R4.2, R4.3, R4.4 y R4.5.

Tabla 5: CU4 Cargar *replay* desde fichero.

CU5	Gestionar velocidad de reproducción *replay*
Actor	Usuarios.
Descripción	Mantener, aumentar o disminuir la velocidad de reproducción del *replay.*
Referencia	R5.1, R5.2 y R5. 3.

Tabla 6: CU5 Gestionar velocidad de reproducción *replay*.

CU6	Gestionar velocidad de retroceso *replay*
Actor	Usuarios.
Descripción	Mantener, aumentar o disminuir la velocidad de retroceso de reproducción del *replay.*
Referencia	R6.1, R6.2 y R6.3.

Tabla 7: CU6 Gestionar velocidad de retroceso *replay*.

CU7	Inicio/ Fin *replay*.
Actor	Usuarios.
Descripción	Saltar al inicio o al final el *replay*.
Referencia	R7.1 y R7.2.

Tabla 8: CU7 Inicio/Fin *replay*.

3.3.3 Diagrama de Casos de Uso del Sistema

En el siguiente diagrama se presenta la relación entre el actor y los casos de uso del sistema.

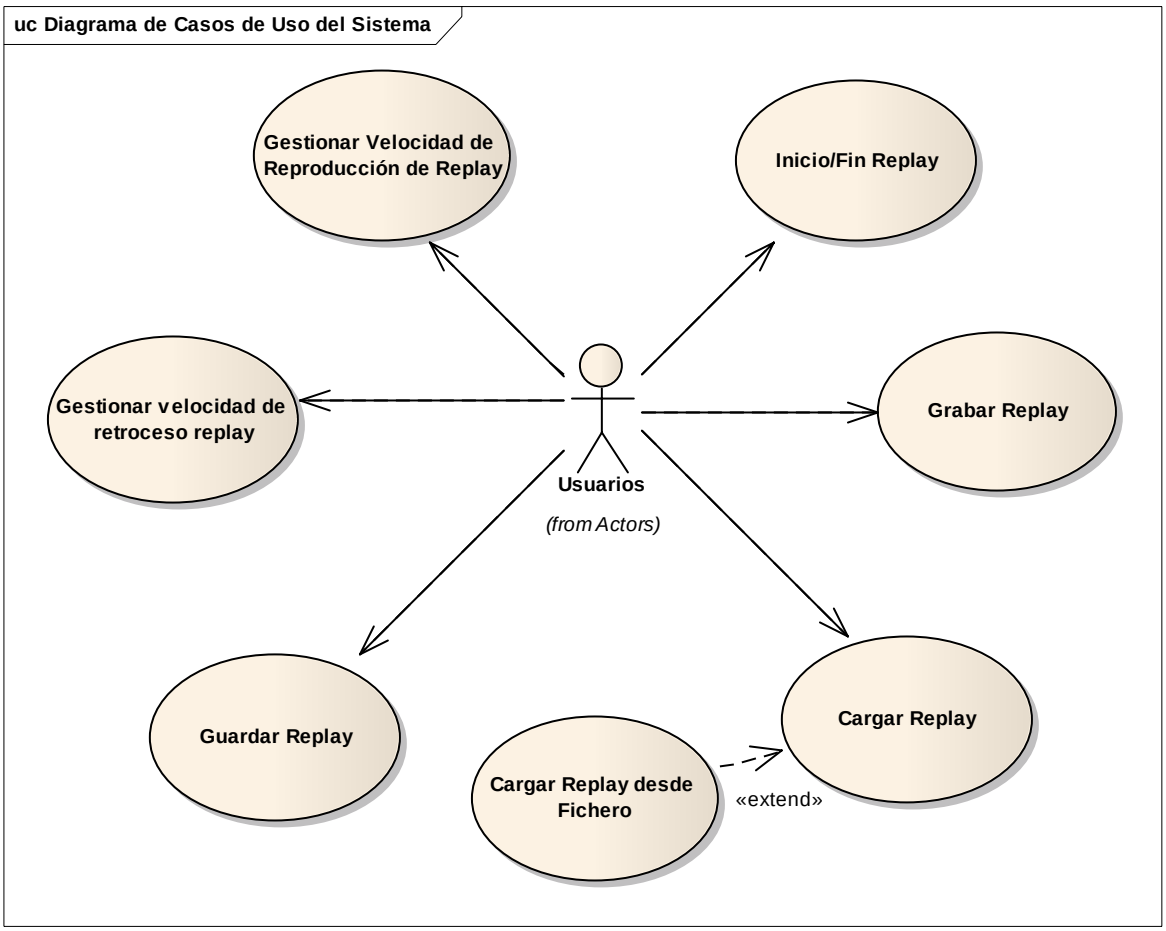

Figura 5: Diagrama de Caso de Uso del Sistema.

3.3.4 Expansión de Casos de Uso

Caso de uso	
CU1	Grabar *replay.*
Propósito	Grabar todos los datos necesarios para poder generar el *replay*.
Actor	Usuarios.
Resumen: El caso de uso se inicia cuando el actor desea comenzar a grabar las acciones que están ocurriendo en el sistema. Una vez iniciada la grabación queda registrado en memoria un grupo de datos.	
Referencias	R1.1, R1.2, R1.3, R1.4 y R1.5.
Precondiciones	Que esté en ejecución la aplicación.
Curso Normal de los Eventos	
Acción del actor	Respuesta del sistema
1-El usuario inicia la grabación.	1.1-Recorre el grafo de escena y almacena en una lista todos los objetos presentes en el mismo.
	1.2-Almacena el estado (posición y orientación) de todos los objetos presentes en la escena en ese momento y el instante de tiempo en que ocurre.
2-Si el usuario oprime alguna tecla que modifique el estado del sistema.	
3-El usuario termina la grabación.	3.1-Almacena el estado final de todos los objetos presentes en la escena.
Curso Alterno de los Eventos	
	2.1-Sigue almacenando los nuevos eventos que generen cambios en el sistema y la variación de

	tiempo en que ocurren.
Poscondiciones	Quedan registrados los datos para la reproducción del *replay*.
Prioridad	Crítico

Tabla 9: Expansión CU1 Grabar *replay*.

Caso de uso		
CU2	Guardar *replay*.	
Propósito	Salvar los datos almacenados durante la grabación.	
Actor	Usuarios.	
Resumen: El caso de uso se inicia cuando el usuario desea salvar los datos almacenados durante la grabación. Los datos son salvados a un fichero.		
Referencias	R2.1, R2.2 y R2.3.	
Precondiciones	Se produjo la grabación del sistema.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
1-El usuario indica salvar la grabación.		1.1-Solicita el nombre para el *replay*.
		1.2-Crea un fichero con este nombre.
		1.3-Guarda la información que fue grabada en el fichero.
Poscondiciones	Quedan registrados en un fichero los datos que fueron almacenados en la lista durante la grabación.	
Prioridad	Secundario.	

Tabla 10: Expansión CU2 Guardar r*eplay*.

Caso de uso		
CU3	Cargar *replay.*	
Propósito	Cargar los datos almacenados durante la grabación.	
Actor	Usuarios.	
Resumen: El caso de uso se inicia cuando el usuario desea cargar los datos almacenados durante la grabación. Se muestra la escena generada por los datos almacenados en el primer instante de tiempo.		
Referencias	R3.1, R3.2.	
Precondiciones	Se produjo la grabación del sistema.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
1-El usuario indica cargar la grabación.		1.1-Carga todos los datos que fueron almacenados durante la grabación.
Poscondiciones	El sistema queda listo para comenzar la reproducción del *replay*.	
Prioridad	Crítico.	

Tabla 11: Expansión CU3 Cargar *replay*.

Caso de uso		
CU4	Cargar *replay* desde fichero.	
Propósito	Cargar los datos de un *replay* almacenado en un fichero.	
Actor	Usuarios.	
Resumen: El caso de uso se inicia cuando el usuario desea cargar un *replay* almacenado en un fichero. Es cargado un *replay* almacenado en un fichero.		
Referencias	R4.1, R4.2, R4.3, R4.4 y R4.5.	
Precondiciones	Existe un fichero.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
1-El usuario indica la opción de cargar un *replay*.		1.1-Muestra todos los *replay* que han sido almacenados en ficheros.
2-El usuario selecciona el *replay* que desea cargar.		1.2-Borra los datos almacenados en memoria. 1.3-Carga los datos del *replay* almacenado en el fichero seleccionado a memoria.
Poscondiciones	Se carga el *replay* almacenado en el fichero y queda listo para su reproducción.	
Prioridad	Secundario.	

Tabla 12: Expansión CU4 Cargar *replay* desde fichero.

Caso de uso		
CU5	Gestionar velocidad de reproducción *replay*	
Propósito	Mantener, aumentar y disminuir la velocidad con que se reproduce el *replay*.	
Actor	Usuarios	
Resumen: El caso de uso se inicia cuando se ha cargado el *replay*. Se reproduce el *replay* a una velocidad igual, mayor o menor a la que fue grabado.		
Referencias	R5.1, R5.2 y R5.3.	
Precondiciones	Se produjo la carga del *replay*.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
		1-Permite realizar las siguientes acciones. a) Si decide reproducir el *replay* a la velocidad normal ver sección “Reproducir *replay*”. b) Si decide reproducir el *replay* a una mayor velocidad ver sección “Aumentar velocidad de *replay*”. c) Si decide reproducir el *replay* a una menor velocidad ver sección “Disminuir velocidad de *replay*”.
Sección “Reproducir *replay*”		
Acción del actor		Respuesta del sistema
1-El usuario selecciona la opción de Reproducir *replay*.		1.1-El sistema reproduce el *replay* a la velocidad que fue grabado.
Sección “Aumentar velocidad de *replay*”		

2-El usuario selecciona la opción de Aumentar velocidad de *replay*.	2.1-El sistema reproduce el *replay* a una velocidad mayor de la que fue grabado.
Sección “Disminuir velocidad de *replay*”	
3-El usuario selecciona la opción de Disminuir velocidad de *replay*.	3.2-El sistema reproduce el *replay* a una velocidad menor de la que fue grabado.
Poscondiciones	Se reproduce el *replay* de acuerdo a los datos cargados y a la velocidad que se seleccione.
Prioridad	Crítico.

Tabla 13: Expansión CU5 Gestionar velocidad de reproducción r*eplay*.

Caso de uso	
CU6	Gestionar velocidad de retroceso *replay.*
Propósito	Mantener, aumentar y disminuir la velocidad de retroceso del *replay*.
Actor	Usuarios.
Resumen: El caso de uso se inicia cuando se ha reproducido el *replay*. A partir del momento deseado se puede retroceder el *replay* manteniendo aumentando o disminuyendo la velocidad de reproducción.	
Referencias	R6.1, R6.2 y R6.3.
Precondiciones	Se produjo la reproducción del *replay*.

Curso Normal de los Eventos	
Acción del actor	Respuesta del sistema
	1-Permite realizar las siguientes acciones. a) Si decide retroceder el *replay* a velocidad normal ver sección “Retroceder *replay*”. b) Si decide retroceder el *replay* a mayor velocidad ver sección “Aumentar velocidad de retroceso del *replay*”. c) Si decide retroceder el *replay* a una menor velocidad ver sección “Disminuir velocidad de retroceso del *replay*”.
Sección “Retroceder *replay*”	
Acción del actor	Respuesta del sistema
1-El usuario selecciona la opción de retroceder el *replay*.	1.1-El sistema retrocede el *replay* a la velocidad que fue grabado.
Sección “Aumentar velocidad de retroceso del *replay*”	
2-El usuario selecciona la opción de	2.1-El sistema retrocede el *replay* a una

Aumentar velocidad de retroceso del *replay*.	velocidad mayor de la que fue grabado.
Sección “Disminuir velocidad de retroceso del *replay*”	
3-El usuario selecciona la opción de Disminuir velocidad de retroceso del *replay*.	3.2-El sistema retrocede el *replay* a una velocidad menor de la que fue grabado.
Poscondiciones	El *replay* es retrocedido a una velocidad igual, mayor o menor de la que fue grabado.
Prioridad	Secundario.

Tabla 14: Expansión CU6 Gestionar velocidad de retroceso *replay*.

CU 7	Inicio/ Fin *replay*.	
Propósito	Saltar al inicio o final del *replay*.	
Actor	Usuarios.	
Resumen: El caso de uso se inicia cuando el usuario desea saltar al inicio o al final del *replay*. Se muestra la reproducción en el lugar que comenzó en caso de ir al inicio o se muestra el final de la reproducción en caso de ir al final.		
Referencias	R7.1 y R7.2.	
Precondiciones	Se produjo la carga del *replay*.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema

1-El usuario indica ir al inicio o al final de la aplicación.	1.1-Permite realizar las siguientes acciones. a) Si decide ir al inicio del *replay* ver sección “Inicio *replay*”. b) Si decide ir al final del *replay* ver sección “Final *replay*”.
Sección “Inicio *replay*”	
Acción del actor	Respuesta del sistema
2-El usuario selecciona la opción de ir al inicio del *replay*.	2.1-El sistema muestra el *replay* en su estado inicial.
Sección “Final *replay*”	
3-El usuario selecciona la opción de ir al final del *replay*.	3.1-El sistema muestra el *replay* en su estado final.
Poscondiciones	El usuario observa el inicio o final del *replay* en dependencia de lo que desee.
Prioridad	Secundario.

Tabla 15: Expansión CU7 Inicio/Fin *replay*.

Conclusiones

Al concluir este capítulo queda definido un modelo de dominio para establecer el contexto del sistema. Quedaron establecidos los requisitos funcionales y no funcionales, descritos los casos de uso y confeccionado el diagrama de casos de uso para representar gráficamente la interacción entre los actores y los casos de uso del sistema así como también la interacción entre los propios casos de uso. Estos resultados reflejan las necesidades cliente además de ser un muy buen punto de partida para el diseño e implementación del sistema.

Capítulo 4: Diseño e Implementación del Sistema

Introducción:

En este capítulo se encuentran el diagrama de clases de diseño del sistema propuesto y los diagramas de secuencia que reflejan la realización de los casos de uso descritos en el capítulo anterior, se incluye además el diagrama de componentes de implementación.

4.1 Estándares de codificación

- **Nombre de ficheros:** Se nombrarán los ficheros .h y .cpp de la siguiente manera: Demo_Name.cpp

- **Clases:** class CClassName;
  Indicando con “C” que es una clase concatenándola con Class y el nombre específico de la clase.

- **Declaración de variables:** Los nombres de variables locales, como los iteradores o los contadores, se especifican en minúscula. Si el nombre de la variable contiene más de una cadena, es decir, es compuesto, comienza con minúscula y luego la próxima cadena con la primera letra en mayúscula y las demás en minúsculas. Siempre y cuando su lectura sea legible.

- **Listas e iteradores STD:**

  std::list<TipodeDato>::iterator itName;
  std::vector<TipodeDato>::iterator itName;
  A los iteradores se le antepone it para especificar que es un iterador.

- **Tipos simples:**

  bool mName;
  int mName;
  unsigned int mName;
  real mName;
  har cName;

- **Métodos:** En el caso de los métodos, siempre empezarán con letras minúscula, de ser un nombre compuesto, a partir de la segunda sub-cadena que lo compone deben empezar con mayúsculas y se le antepondrá el identificador del tipo de dato de devolución. Solamente los constructores y destructores tendrán el mismo nombre de la clase. Y a los parámetros que son pasados a los métodos se le antepondrá una "p" minúscula y las demás sub-cadenas que componen el nombre del parámetro comenzarán con mayúsculas.

- **Ejemplo:**
  void removeAllRecordableObjects();
  oid setKeyPressedObject1(bool pValue);

- **Comentarios:** Utilizar el estándar // sólo para comentarios de una sola línea en caso de que sean varias utilizar el estándar /* */.

4.2 Características de Ogre

Ogre es un motor gráfico de código libre. Está complementado con CEGUI y OIS. Estas dos bibliotecas no forman parte este motor, pero sus desarrolladores las han usado para poder complementarlo.

La primera, es una interfaz gráfica de usuario que permite crear diálogos, cajas de edición de texto, cajas de lista, etc, la segunda es una biblioteca de interfaz de usuario que permite capturar y manejar los eventos de teclado, ratón y joystick. Destacable de este motor es su facilidad inicial de uso, aunque se requiere un nivel medio-alto de C++ para poder emplearlo de manera satisfactoria.

Ogre se presenta en dos paquetes, un SDK, listo para desarrollar, muy recomendable para novatos, y otro en formato código fuente, mucho más propio de usuarios avanzados y que permite, por ejemplo, depuración dentro del código de Ogre.

Se complementa además, con muchas herramientas de exportación de editores gráficos 3D, como Blender, 3D Studio Max, Maya, etc. y un paquete de demos, en los cuales se pueden apreciar las magníficas capacidades de este motor (se tiene disponible sus código fuentes que puede servir de aprendizaje).

También cuenta con una arquitectura basada en plugins, que lo hace ser muy versátil. Para la plataforma Windows se tiene dos módulos de *render*, uno para DirectX y otro para OpenGL para Linux, se cuenta con soporte OpenGL.

4.3 Características de la biblioteca TinyXML

TinyXML es un analizador que integra C++ con XML (*eXtensible Markup Languaje*) y se puede acoplar fácilmente en otros programas. TinyXML analiza un documento XML, construido a partir de un *Document Object Model* (DOM) que puede ser leído, modificado y guardado. Tiene diferentes formas de acceso e interacción con los datos XML. TinyXML es diseñado para ser rápido y fácil de aprender. Solamente tiene dos archivos .h y cuatros .cpp. Basta con añadir estos al proyecto y puede comenzar con su uso. Es un *software* libre distribuido bajo los términos de la licencia zlib/libpng.

4.4 Diagramas de Clases de diseño

En el diagrama de la página siguiente se muestra la relación entre las clases de diseño, nótese que refleja además algunas clases implementadas ya en Ogre y TinyXML, pero sin las cuales sería muy complejo entender la arquitectura del sistema. En lo adelante solo se desarrollarán los artefactos de nuestro sistema, aunque se mencionen o representen elementos de Ogre y TinyXml que son arquitectónicamente significativos.

4.4.1 Diseño de clases por paquete

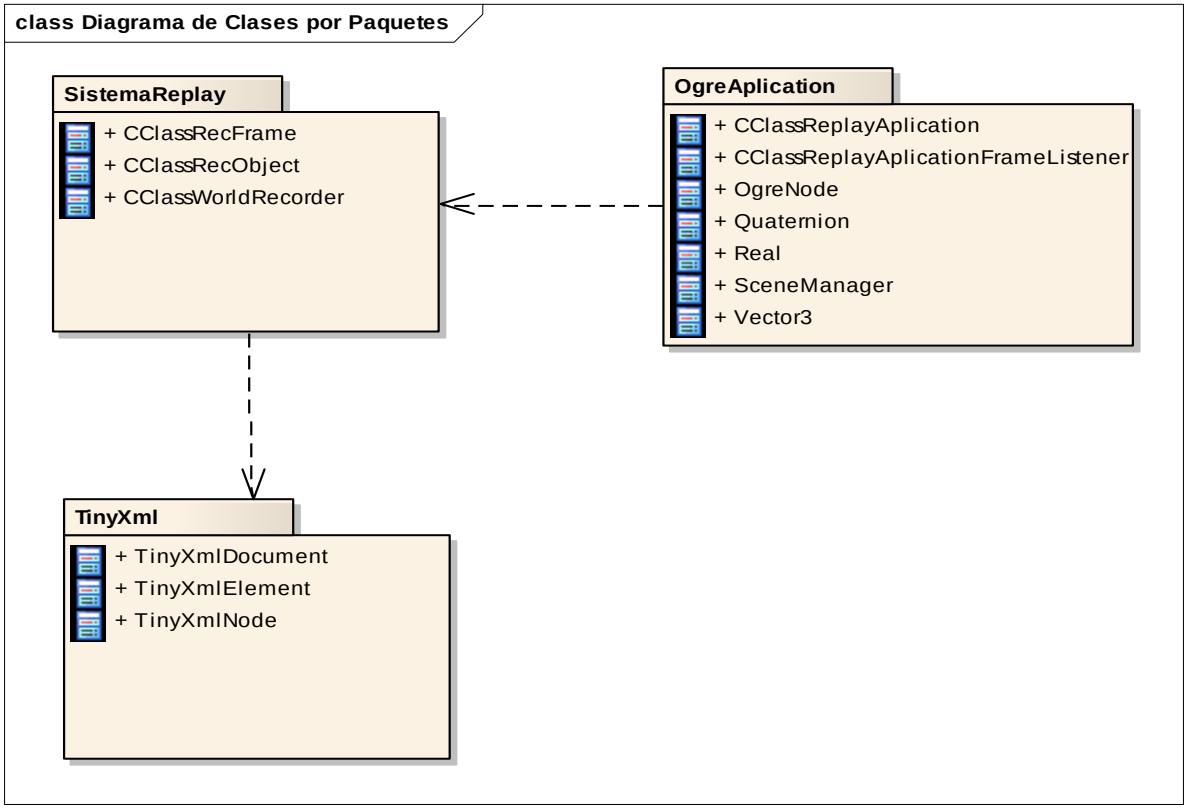

Figura 6: Diagrama de clases de diseño por paquetes.

4.4.2 Diseño de clases paquete SistemaReplay

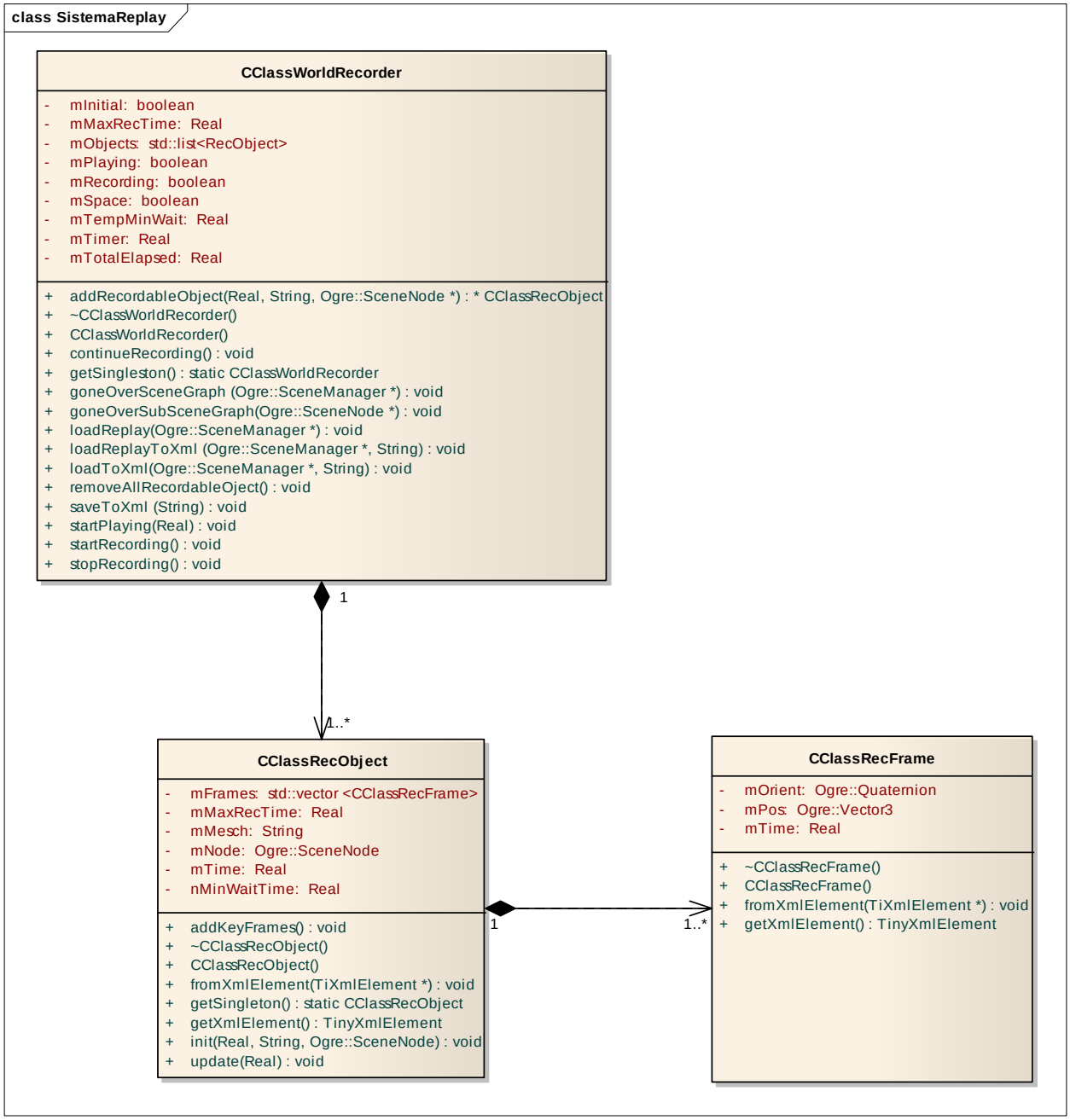

Figura 7: Diagrama de clases del paquete SistemaReplay.

4.4.3 Diseño de clases paquete TinyXml

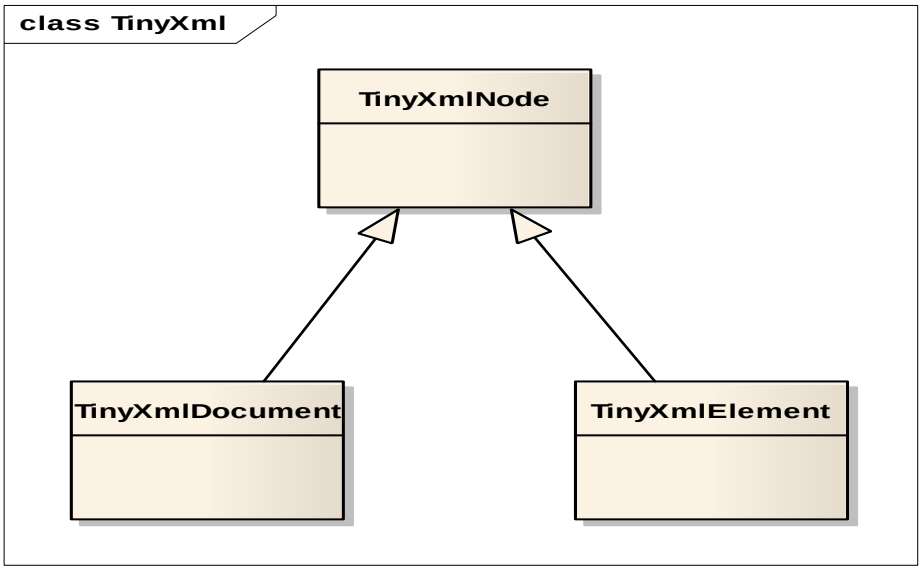

Figura 8: Diagrama de clases del paquete TinyXml.

4.4.4 Diseño de clases paquete OgreAplication

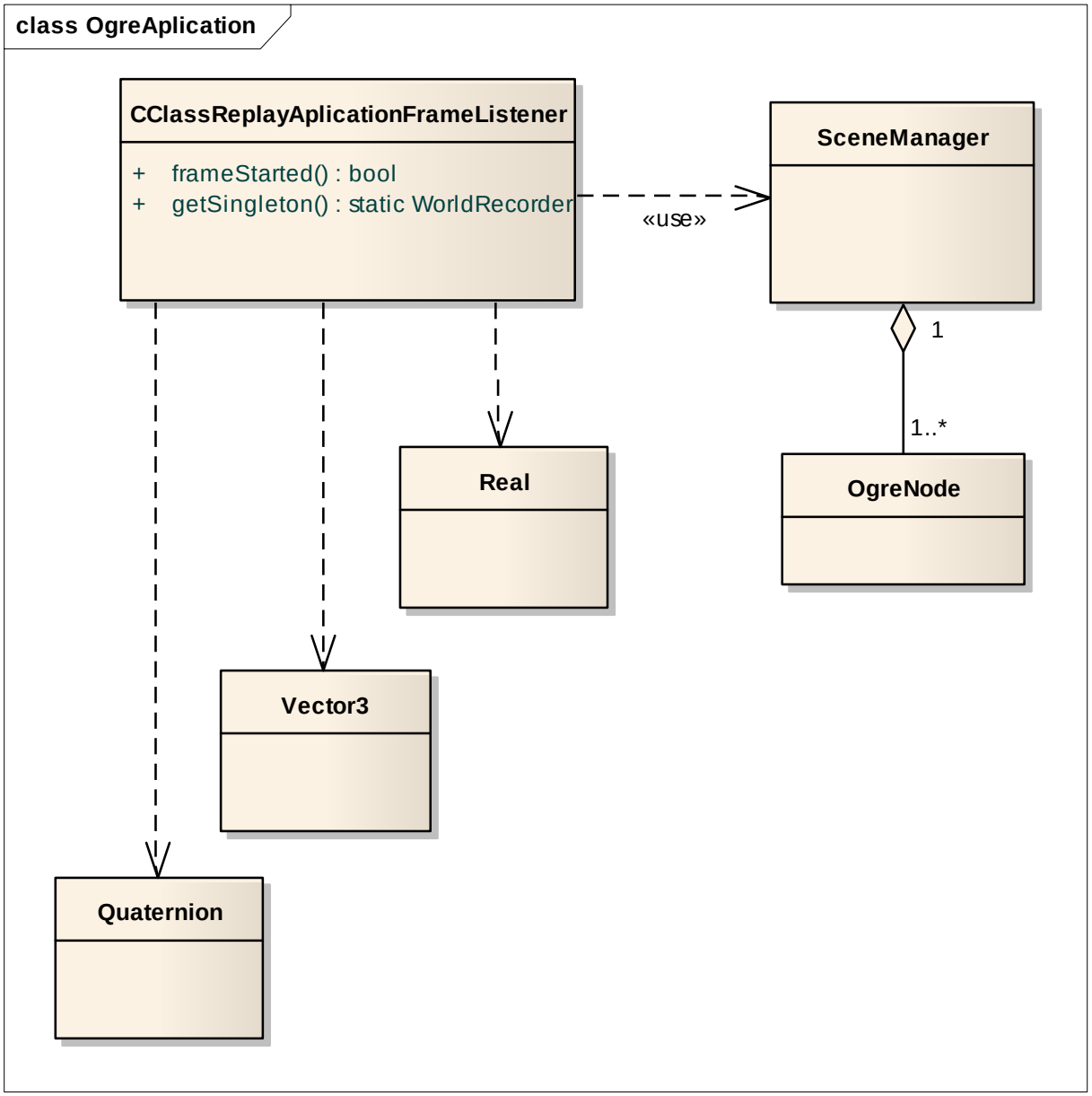

Figura 9: Diagrama de clases del paquete OgreAplication.

4.4.5 Diseño de clases General.

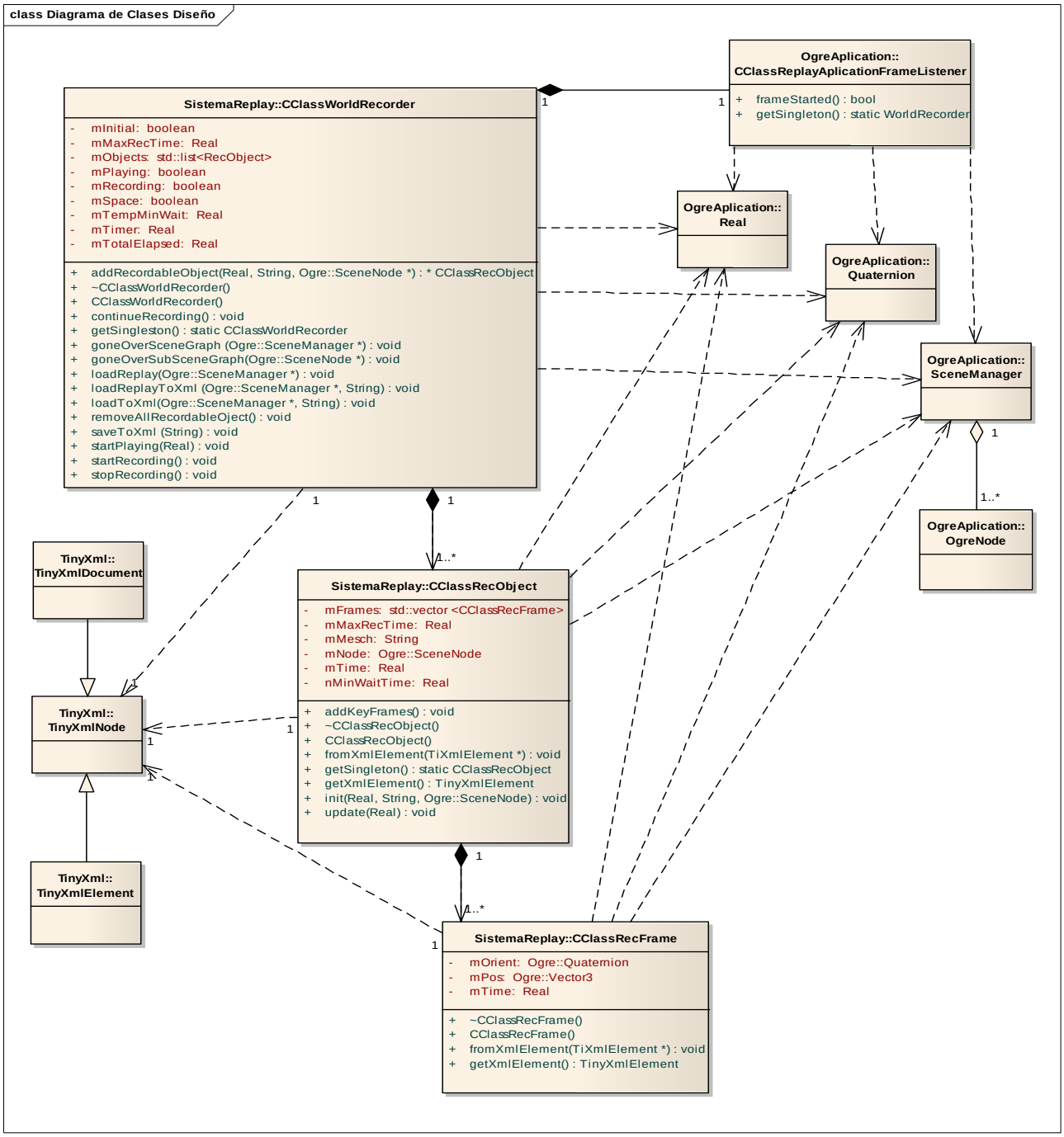

Figura 10: Diagrama de clases General.

4.5 Descripción de las clases de diseño

Nombre: CClassWorldRecorder	
Tipo de clase: Contenedora	
Atributo: mMaxRecTime	Tipo: Real
Atributo: mTotalElapsed	Tipo: Real
Atributo: mTempMinWait	Tipo: Real
Atributo: mObjects	Tipo: std::list<CClasRecObject>
Atributo: mRecording	Tipo: bool
Atributo: mPlaying	Tipo: bool
Atributo: mSpace	Tipo: bool
Atributo: mInitial	Tipo: bool
Atributo: mTimer	Tipo: Real
Para cada responsabilidad:	
Nombre:	addRecordableObject(Ogre::SceneNode *pNode,String pMesh,Real pMinWait)
Responsabilidad:	Inicializa y luego adiciona todos los RecObjects en la lista.
Nombre:	removeAllKeyframes()
Responsabilidad:	Borra todos los RecFrame que están almacenados.
Nombre:	removeAllRecordableObjects()
Responsabilidad:	Borra todos los RecObjects que están almacenados.
Nombre:	startRecording()

Responsabilidad:	Comienza la grabación.
Nombre:	continueRecording()
Responsabilidad:	Continua la grabación.
Nombre:	stopRecording()
Responsabilidad:	Para la grabación.
Nombre:	startPlaying(Real pTimeUpdate)
Responsabilidad:	Comienza la reproducción del *replay.*
Nombre:	saveToXML(string pFilename)
Responsabilidad:	Salva para el fichero XML la información almacenada.
Nombre:	loadToXML(String pFilename, Ogre::SceneManager* mSceneMgr)
Responsabilidad:	Restaura los elementos de la escena a partir de los grabados en el XML.
Nombre:	loadReplay(Ogre::SceneManager * pSceneMgr)
Responsabilidad:	Carga el estado inicial de todos los objetos almacenados en memoria.
Nombre:	loadReplayToXML(String pFilename, Ogre::SceneManager * pSceneMgr)
Responsabilidad:	Carga el estado inicial de todos los objetos almacenados en el fichero.
Nombre:	goneOverSceneGraph(Ogre::SceneManager * mSceneMgr)
Responsabilidad:	Recorre el grafo de escena.

Tabla 16: Descripción de la clase CClassWorldRecorder.

Nombre: CClassRecObjects	
Tipo de clase: Controladora	
Atributo: mNode	Tipo: Ogre::SceneNode
Atributo: mLastTime	Tipo: Real
Atributo: mLastKeyframe	Tipo: unsigned int
Atributo: mFrames	Tipo: std::vector<CClassRecFrame>
Atributo: mMesh	Tipo: String
Atributo: mTime	Tipo: Real
Atributo: mMinWaitTime	Tipo: Real
Atributo: mMaxRecTime	Tipo: Real
Para cada responsabilidad	
Nombre:	getXMLElements()
Responsabilidad:	Crea nodo de tipo TinyXmlElement y lo retorna.
Nombre:	fromXmlElement(const TiXmlElement* pElem)
Responsabilidad:	Lee el nodo TinyXml y actualiza los atributos de la clase correspondiente.
Nombre:	Init(Ogre::SceneNode *pNode,String pMesh,Real pMinwait)
Responsabilidad:	Inicializa los parámetros y reserva memoria para los

	RecFrames.
Nombre:	getSingleton()
Responsabilidad:	Crea una sola instancia de su clase en toda la aplicación.
Nombre:	addRecFrame()
Responsabilidad:	Adiciona un objeto RecFrame a la lista de RecFrame.
Nombre:	removeAllKeyFrame()
Responsabilidad:	Borra todos los objetos KeyFrame de la lista de KeyFrame.
Nombre:	update(Real pTime)
Responsabilidad:	Toma los RecFrame almacenados en memorias y actualiza los objetos en la escena.

Tabla 17: Descripción de la clase CClassRecObject

Nombre: CClassRecFrame	
Tipo de clase: Entidad	
Atributo: mPos	Tipo: Ogre::Vector3
Atributo: mOrient	Tipo: Ogre::Quaternion
Atributo: mTime	Tipo: Real
Para cada responsabilidad:	
Nombre:	getXmlElements()
Responsabilidad:	Crea nodo de tipo TinyXmlElement y lo retorna.
Nombre:	fromXMLElement(const TiXmlElement* pElem)
Responsabilidad:	Lee el nodo TinyXml y actualiza los atributos de la clase correspondiente.

Tabla 18: Descripción de la clase CClassRecFrame

4.6 Diagramas de secuencia

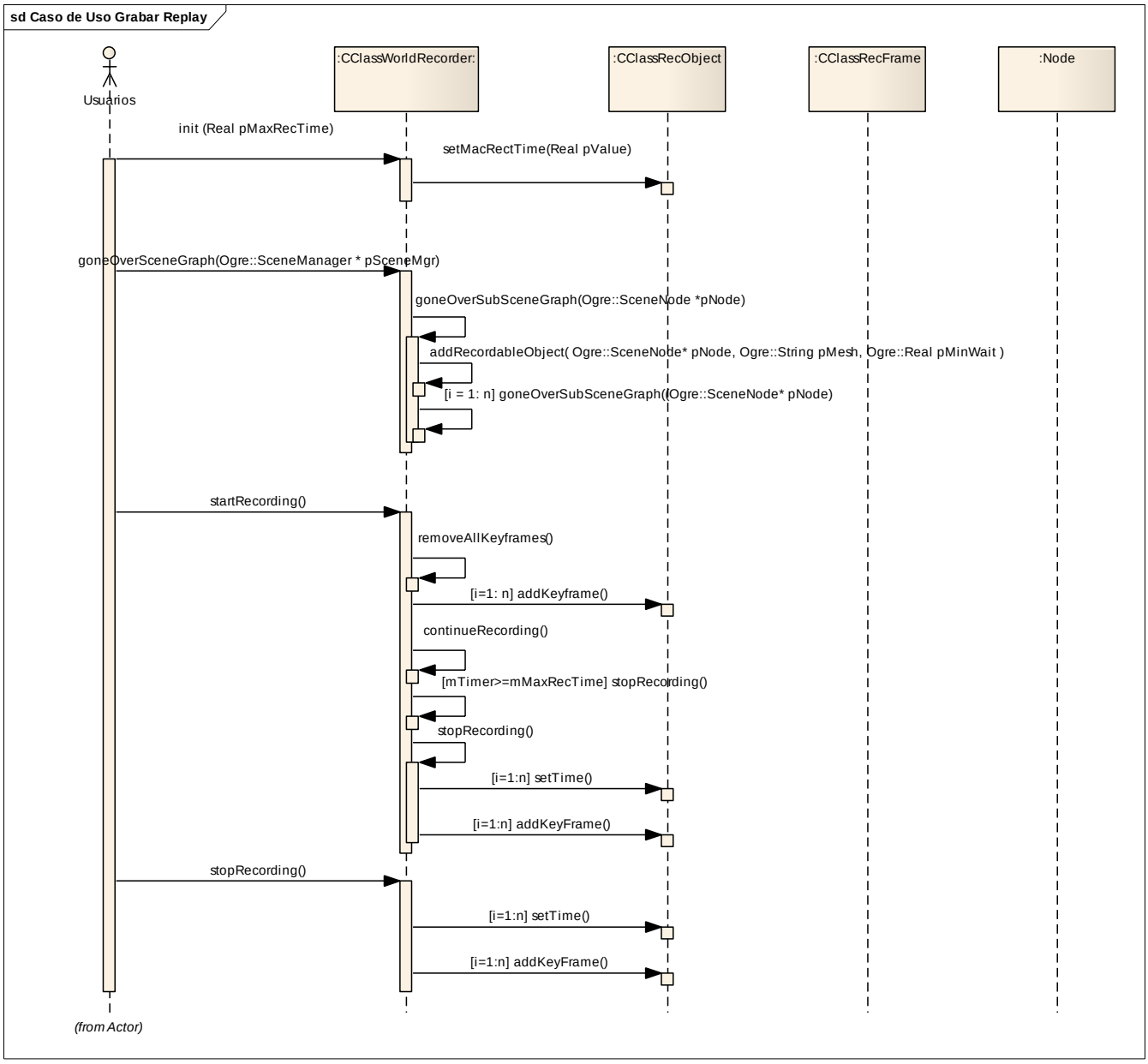

Figura 11: Diagrama de secuencia "Grabar *replay*".

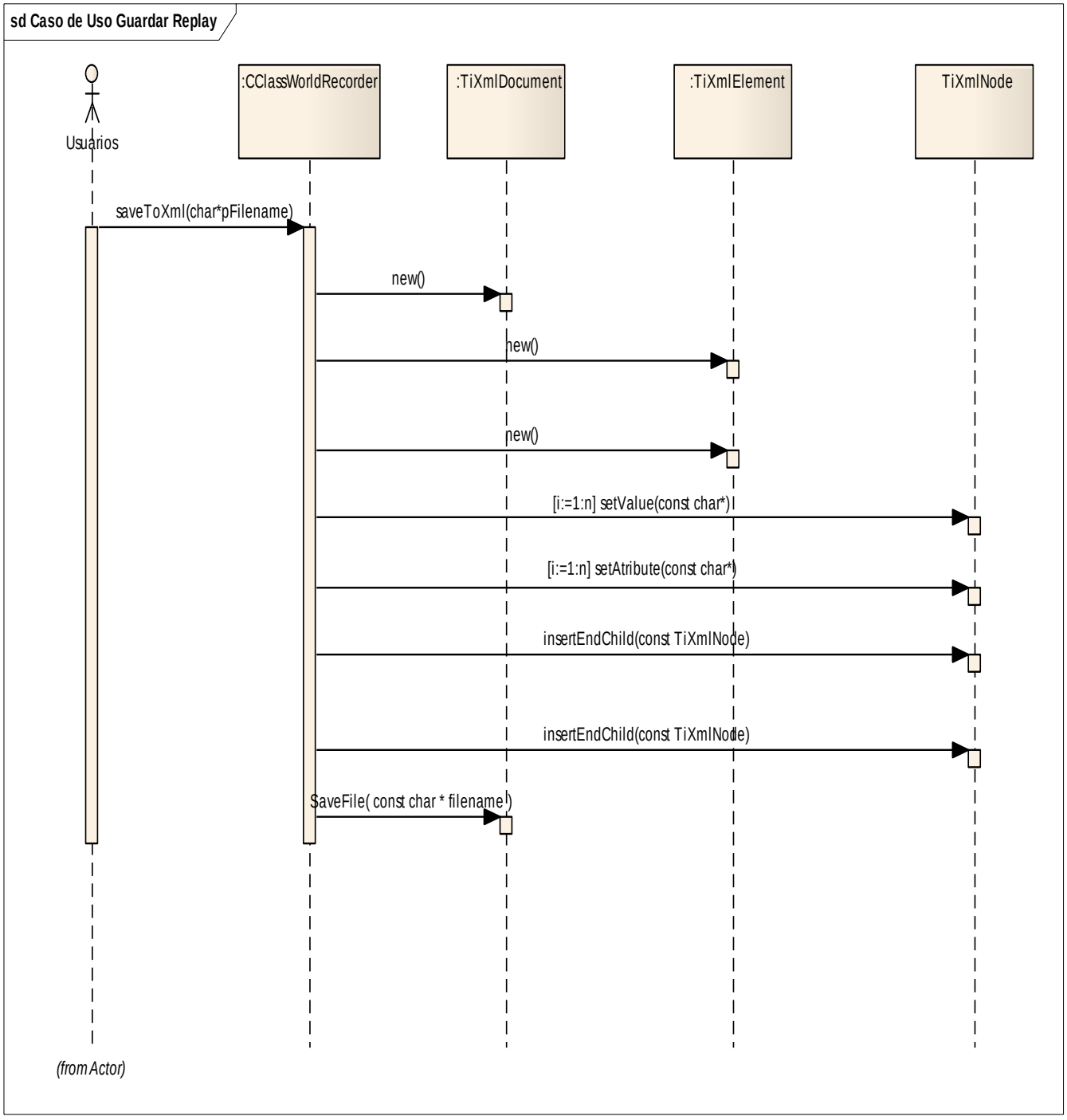

Figura 12: Diagrama de secuencia "Guardar r*eplay*".

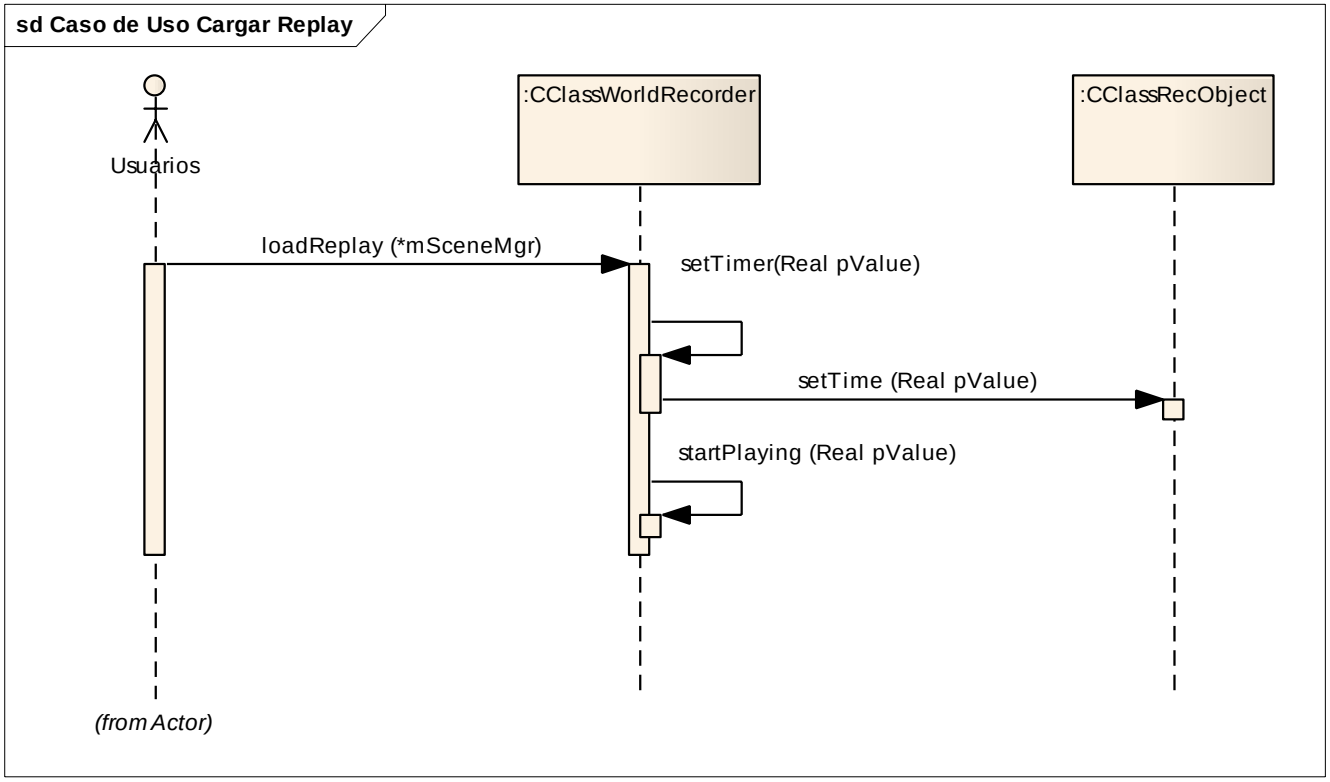

Figura 13: Diagrama de secuencia "Cargar *replay*".

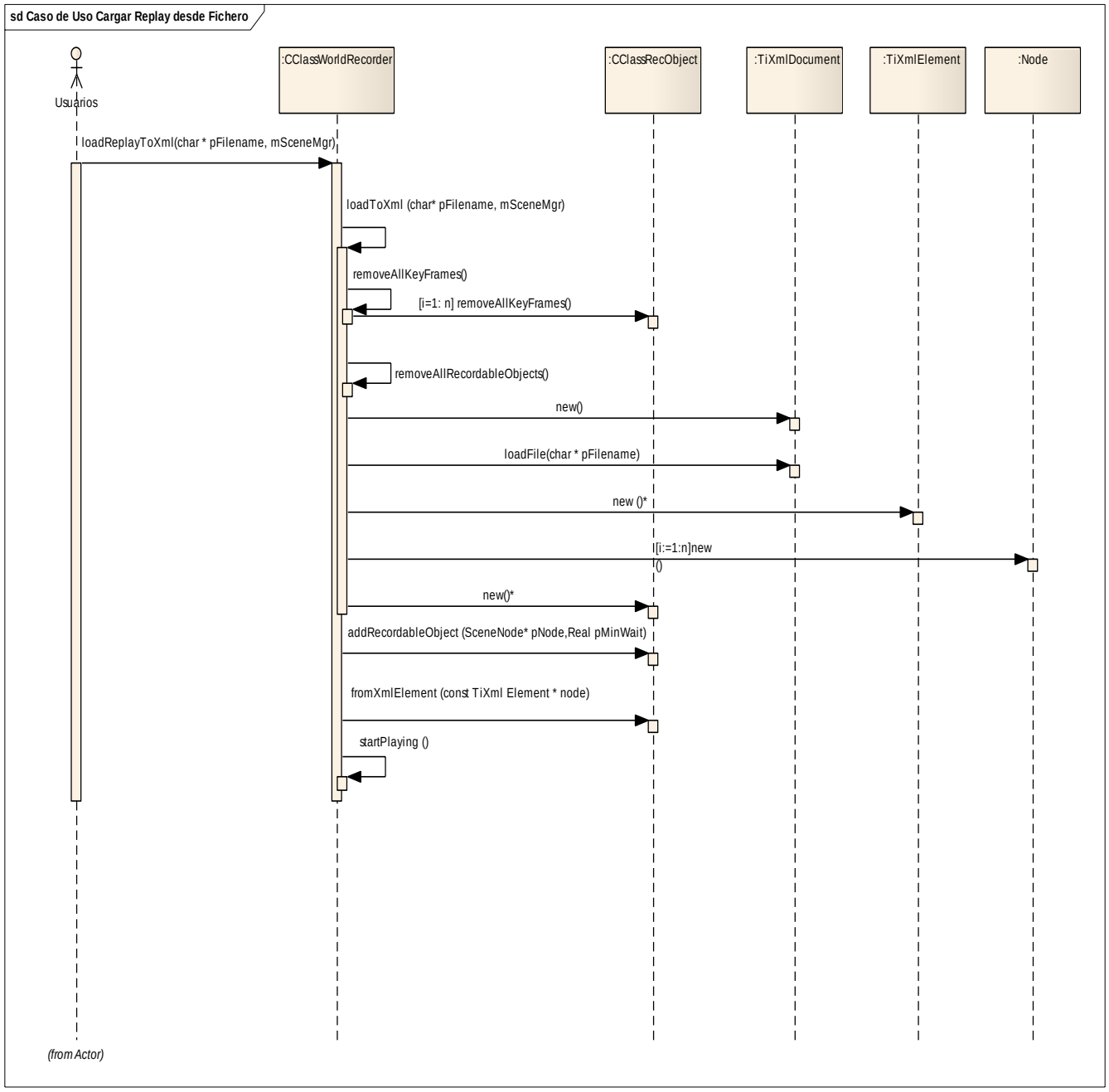

Figura 14: Diagrama de secuencia "Cargar *replay* de fichero".

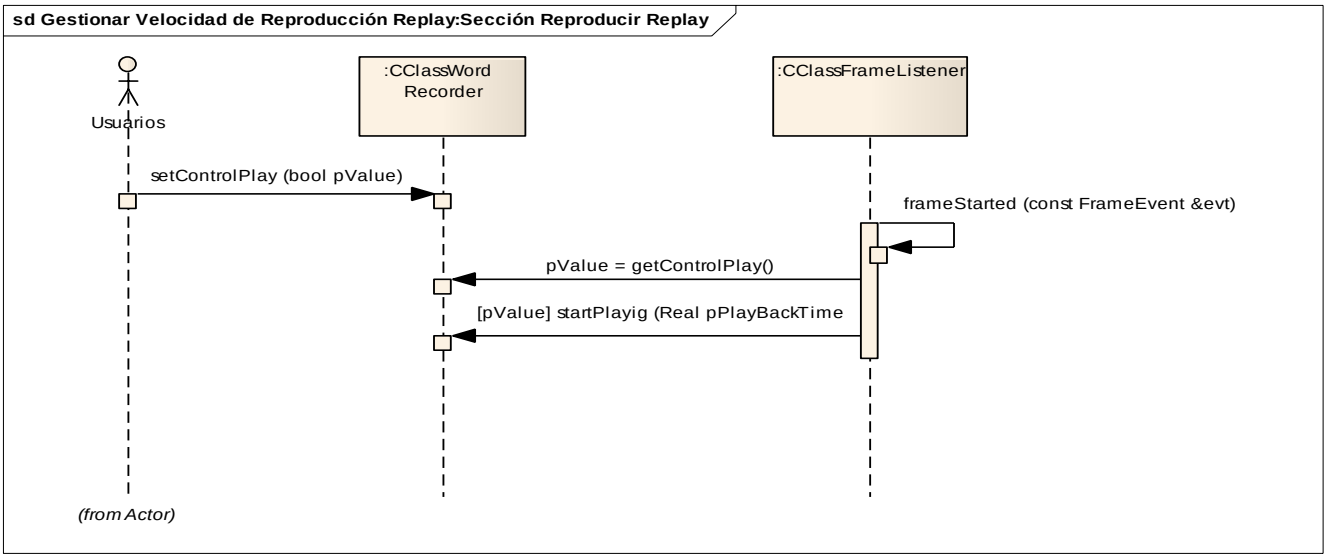

Figura 15: Diagrama de secuencia "Gestionar velocidad de reproducción *replay*". Sección: "Reproducir *replay*".

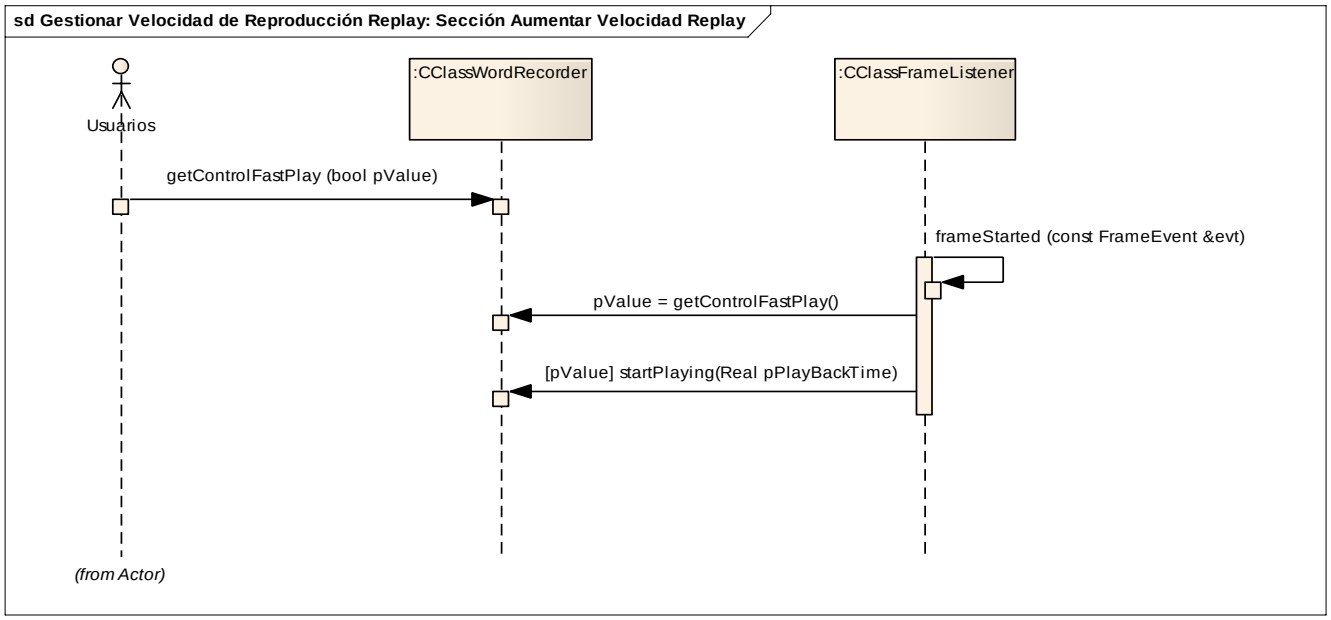

Figura 16: Diagrama de secuencia "Gestionar velocidad de reproducción *replay*". Sección: "Aumentar velocidad r*eplay*".

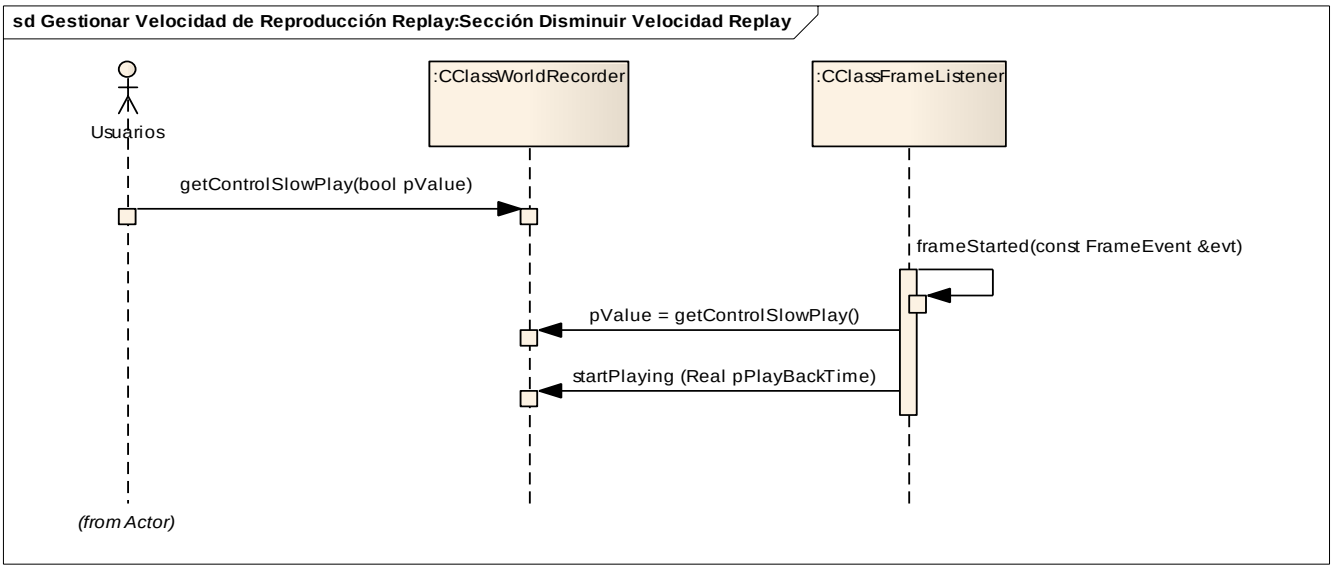

Figura 17: Diagrama de secuencia "Gestionar velocidad de reproducción *replay*". Sección: "Disminuir Velocidad *replay*".

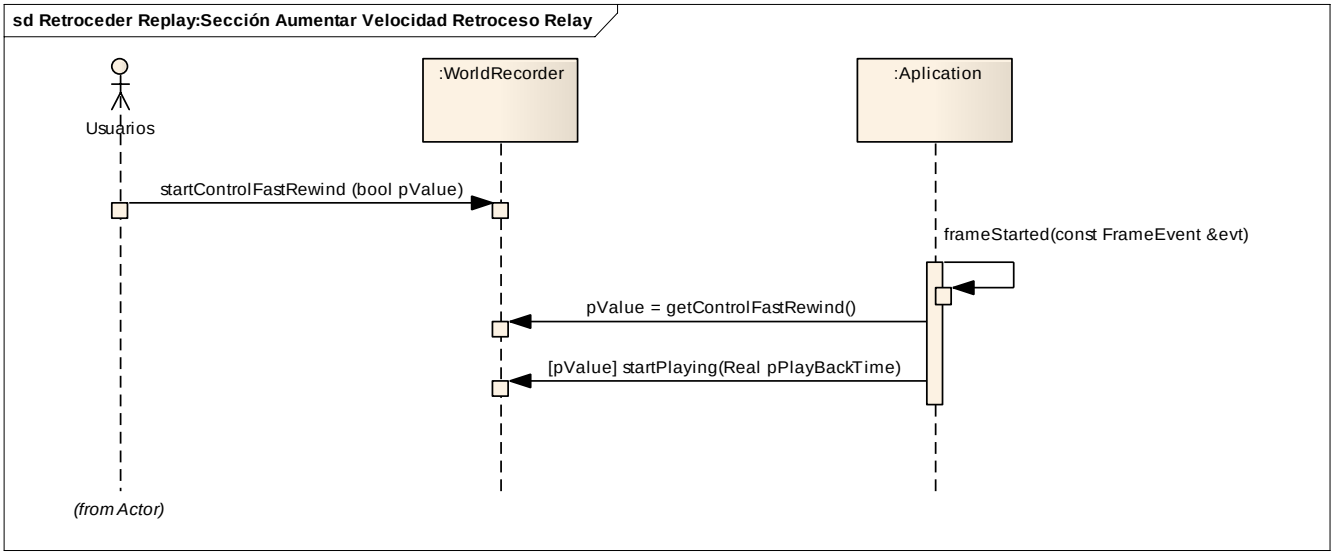

Figura 16: Diagrama de secuencia "Gestionar velocidad de reproducción". Sección: "Aumentar velocidad reproducción *replay*".

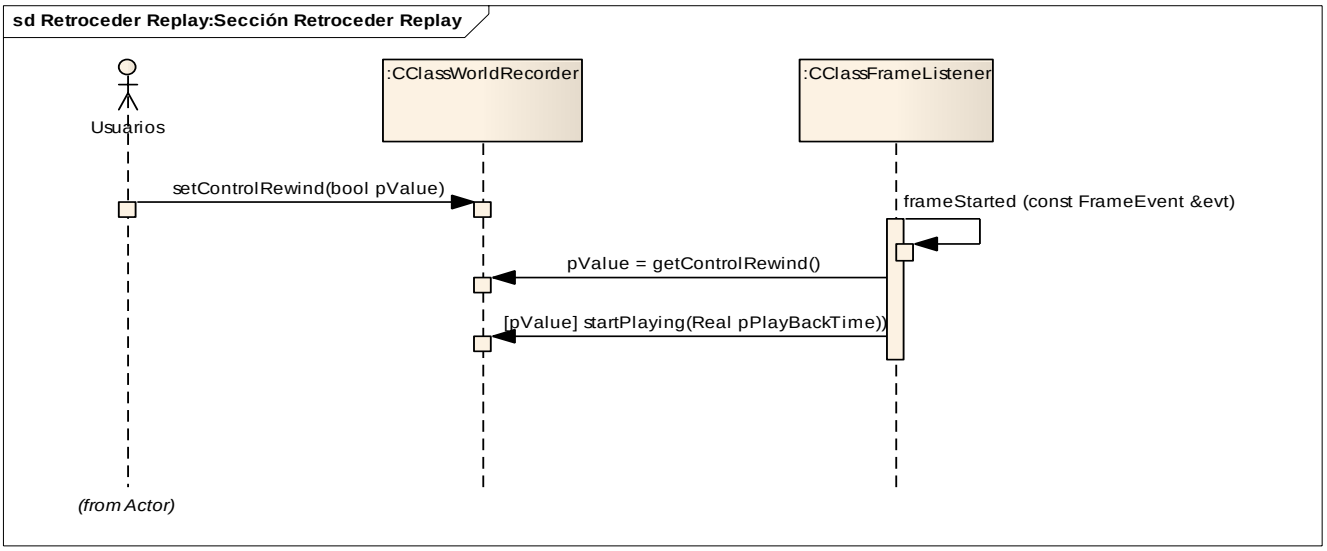

Figura 17: Diagrama de secuencia "Retroceder *replay*". Sección: "Retroceder *replay*".

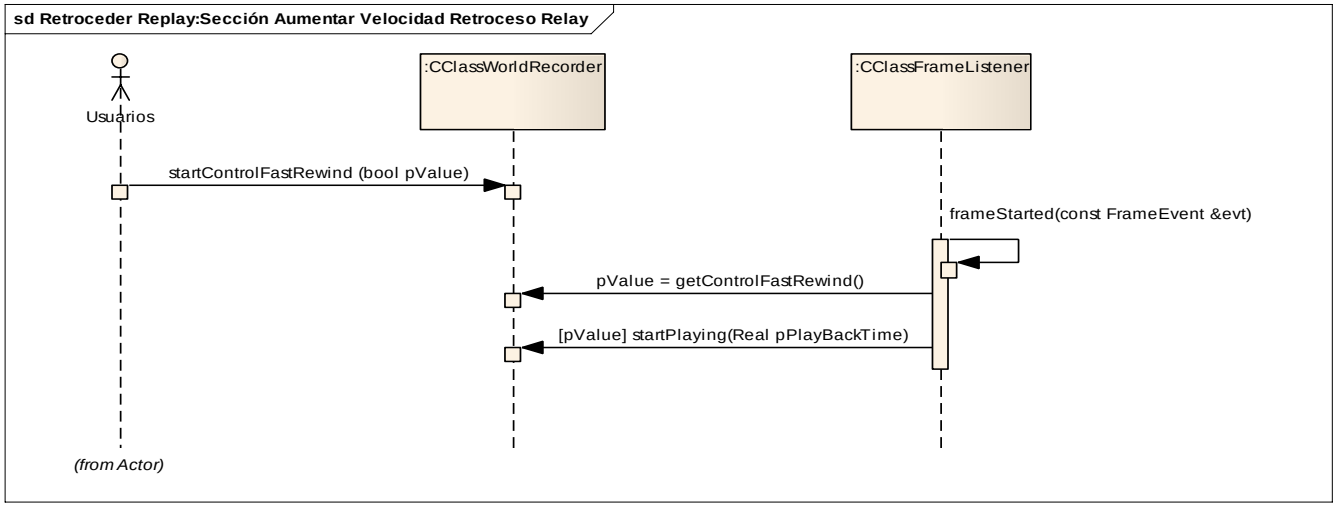

Figura 18: Diagrama de secuencia "Retroceder *replay*". Sección: "Aumentar velocidad retroceso r*eplay*".

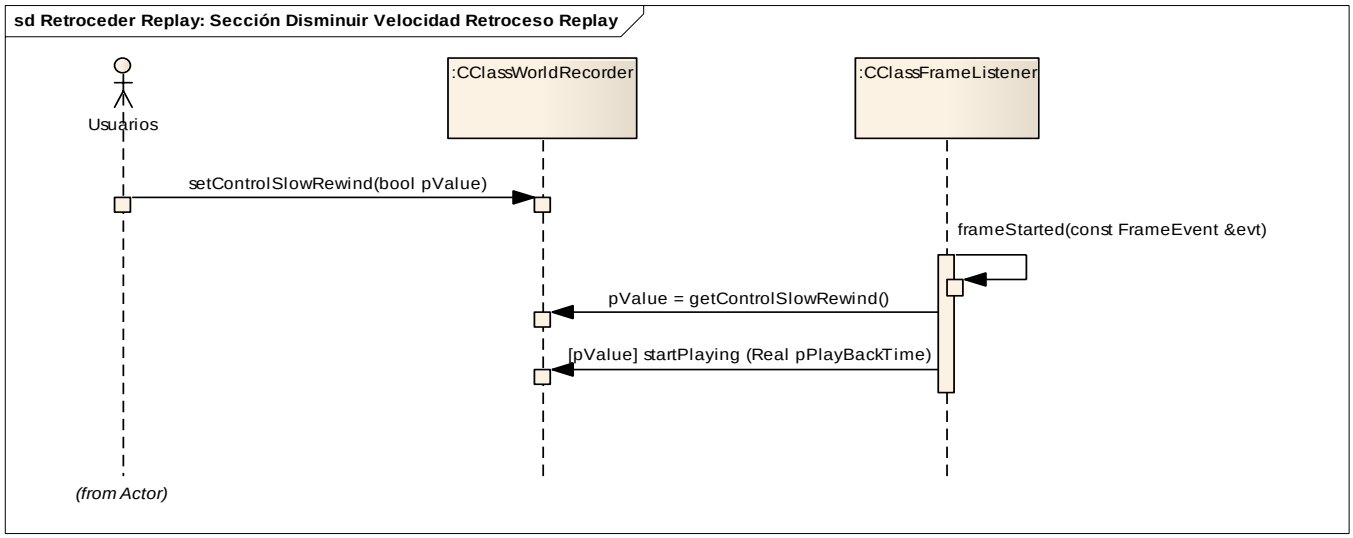

Figura 19: Diagrama de secuencia "Retroceder *replay*". Sección: "Disminuir velocidad retroceso r*eplay*".

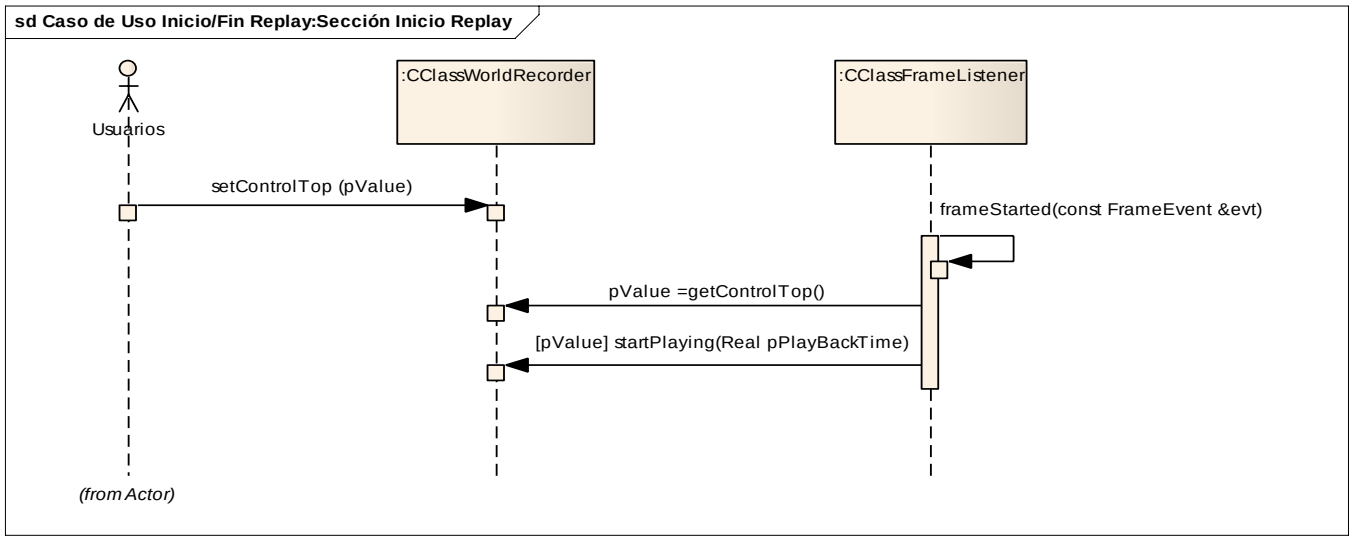

Figura 20: Diagrama de secuencia "Inicio/Fin r*eplay*". Sección: "Inicio *replay*".

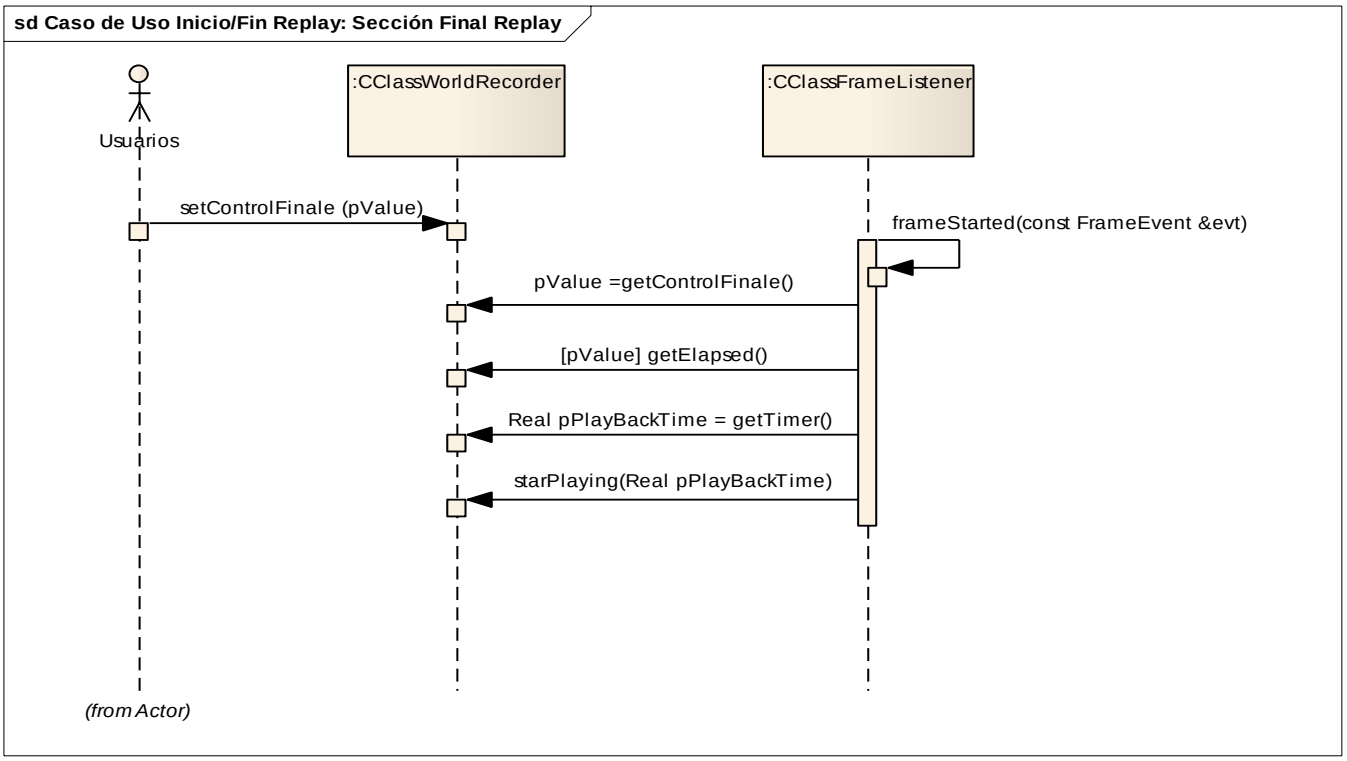

Figura 21: Diagrama de secuencia "Inicio/Fin r*eplay*". Sección: "Final r*eplay*".

4.7 Diagrama de Componentes

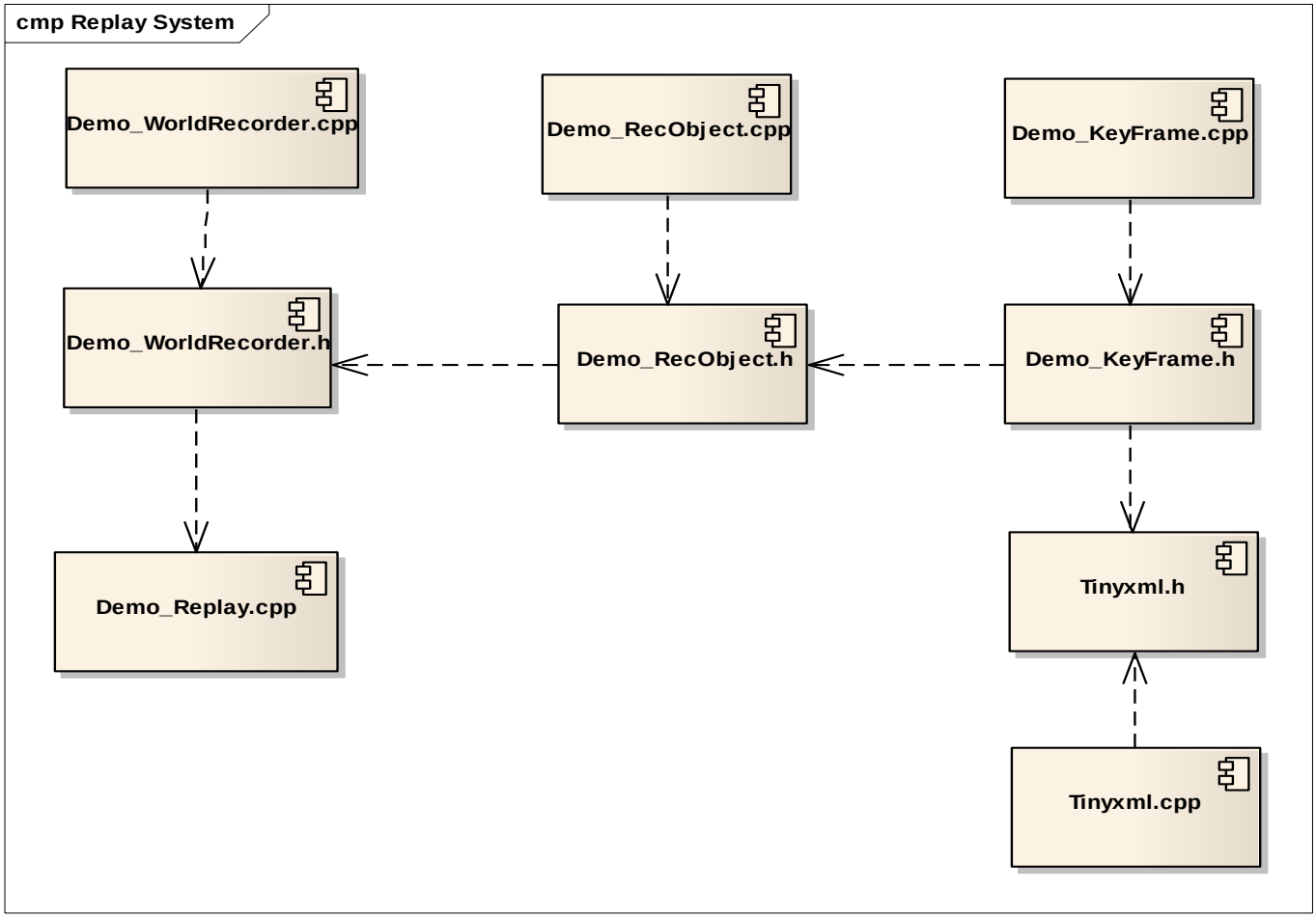

Figura 22: Diagrama de componentes sistema de *replay*.

Conclusiones

A lo largo del este capítulo se mostraron los artefactos necesarios para el desarrollo del un sistema de *replay*, pertenecientes a las etapas de diseño e implementación. Fundamentalmente se definió una arquitectura compatible con Ogre y la biblioteca TinyXML para la realización de este sistema de *replay* y la mensajería entre las clases en forma secuencial.

Después de ser concebido y detallado el diseño se pasó a detallar cómo se harán físicas las clases de diseño, consolidando el proceso de desarrollo para pasar a la programación de los casos de uso correspondientes al primer ciclo de desarrollo.

Conclusiones Generales

Para el cumplimiento de los objetivos de este trabajo de diploma en función con las necesidades del cliente, fue indispensable un estudio de los enfoques de *replay* existentes, así como sus ventajas y desventajas en aras de brindar una solución eficiente.

Finalmente se propone una solución con un enfoque determinista factible para ser usado en SRV, acotándose siempre la solución a las especificidades del sistema y con previa antelación, tener concebidos que se le quiera adicionar la funcionalidad de repetición. La solución queda descrita en términos de los flujos de trabajo de requerimientos, diseño e implementación establecidos por RUP.

Recomendaciones

Aplicar algún método de compresión al fichero que se genera para salvar los datos del *replay*.

Aplicar el mismo enfoque desarrollado en el trabajo a sistemas que tengan elementos como: física, redes, sonido e inteligencia artificial.

Referencias Bibliográficas

[1] W. R. Sherman, A. Craig. “Understanding Virtual Reality: Interface, Application, and Design”, Morgan Kaufmann, 2003.

[2] Colombia aprende “Simtor RV. Sistema se Simulación basado en Realidad Virtual de Tornos metálicos utilizados en Metalmecánica” 1998.

http://www.colombiaaprende.edu.co/html/mediateca/1607/articles-74628_archivo.pdf

Juan Carlos Sepulpeda y Juan Manuel Escadón Pérez.

[En Línea] [Citado el: 25 de 10 de 2008].

[3]SceneToolKit Documentación Para usuarios de la Herramienta Versión 8.05. “Newton” Mayo 2008.

[4] Pro OGRE 3D Programming. Gregory Junker 2006.

Canalsocial "Determinismo Científico" 1991.
http://www.canalsocial.net/GER/ficha_GER.asp?id=4245&cat=ciencia

Roberto Saumells.

[En Línea] [Citado el: 25 de 10 de 2008].

[6] Definición “Definición de determinismo”. http://www.definicion.org/determinismo

[En Línea] [Citado el: 25 de 10 de 2008].

[7] Theorie Analytique des Probabilities. Marquis de Laplace 1820.

[8] Oposipedia “La construcción científica de la realidad. Determinismo e indeterminismo. El postulado de la objetividad” 1993.
http://www.oposinet.com/filosofia/temas/oposiciones_filosofia_T10.php

[En Línea] [Citado el: 25 de 10 de 2008].

[9] Gamasutra “Instant Replay : Building a Game Engine with Reproducible Behavior” Julio 13 2001. http://www.gamasutra.com/view/feature/3057/instant_replay_building_a_game_.php

Patrick Dickinson.

[En Línea] [Citado el: 2 de 11 de 2008].

[10] Game Programming Gems II, Mark Deloura Agosto 2002.

[11] AI Game Programming WISDOM. Paul Tozour, Ion Storm Austin 2002.

[12] Gamasutra “Developing Your Own Replay System” febrero 2004.

http://www.gamasutra.com/view/feature/2029/developing_your_own_replay_system.php

Cyrille Wagner.

[En Línea] [Citado el: 2 de 11 de 2008].

[13] ogre3d “about”. http://www.ogre3d.org/about

[En Línea] [Citado el: 30 de 10 de 2008].

[14] Utah Engineering “The Ultimate Display” 1965.

http://www.eng.utah.edu/~cs6360/Readings/UltimateDisplay.pdf

[En Línea] [Citado el: 25 de 10 de 2008].

[15] PCMAG.COM “Graphic engine definition from PC Magazine Encyclopedia”

http://www.pcmag.com/encyclopedia_term/0,2542,t=graphics+engine&i=43927,00.asp

[En Línea] [Citado el: 2 de 11 de 2008].

[16] Marks s, Windsor J, Wunsche B.” Evaluation of Game Engines for Simulated Clinical Training” 2007.

[17] Universidad de Valencia “Motores Gráficos” 2007.

http://informatica.uv.es/iiguia/IG/motores_graf.pps

Rubén Talón Argente.

[En Línea] [Citado el: 2 de 11 de 2008].

Glosario de Términos

Multiplataforma: Término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

Biblioteca: En Inglés library, en términos informáticos, refiere al conjunto de rutinas que realizan las operaciones usualmente requeridas por los programas. Las bibliotecas pueden ser compartidas, lo que quiere decir que las rutinas de la biblioteca residen en un fichero distinto de los programas que las utilizan. Los programas enlazados con bibliotecas compartidas no funcionarán a menos que se instalen las bibliotecas o librerías necesarias.

C++: Lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (POO). C++ está considerado por muchos como uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel.

Hardware: Componentes físicos de una computadora o de una red (a diferencia de los programas o elementos lógicos que los hacen funcionar).

Realidad Virtual: Simulación generada por computadora de imágenes o ambientes tridimensionales interactivos con cierto grado de realismo físico o visual.

Render: Proceso de obtención de imágenes por computadora.

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

Virtual: Término utilizado para hacer referencia a algo que no tiene existencia física o real, sólo aparente.

Tiempo Real: Término usado en el mundo de los gráficos por computadora para las aplicaciones interactivas con respuesta en intervalos de tiempo que parecen instantáneos al usuario, usualmente más de 16 fotogramas por segundos (fps).