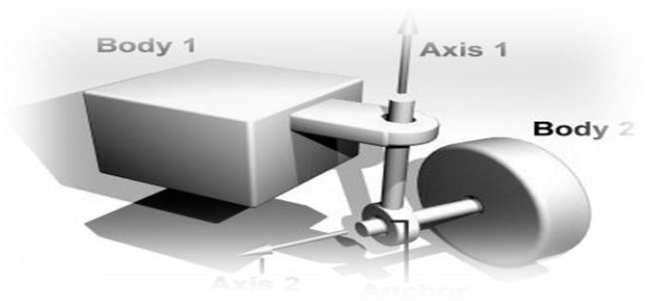
Universidad de las Ciencias Informáticas

Facultad 5 Entornos Virtuales

Título: Edición de sistemas articulados de cuerpos rígidos para las animaciones en los videojuegos.



Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autores: Hassán Lombera Rodríguez
Alberto Eliseo Pacheco Allende

Tutores: Ing. Igr Alexánder Fernández Saúco
Lic. Eduardo Lago Aguilar

Julio 2008

"En igualdad de condiciones la solución más sencilla es probablemente la correcta."

Navaja de Ockham.

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente el día 1ro de julio del año 2008.

_____
Hassán Lombera Rodríguez
Autor

_____
Alberto Eliseo Pacheco Allende
Autor

_____
Lic. Eduardo Lago Aguilar
Tutor

_____
Ing. Igr Alexánder Fernández Saúco
Tutor

DATOS DE CONTACTO DE LOS TUTORES

Nombre y Apellidos: *Eduardo Lago Aguilar*

Edad: *27*

Ciudadanía: *Cubana*

Institución: *DATYS*

Título: *Licenciado en Ciencias de la Computación*

Categoría Docente: *Profesor Instructor*

E-mail: *eduardo.lago@datys.co.cu*

Ocupación Actual: *Especialista Principal*

Nombre y Apellidos: *Igr Alexánder Fernández Saúco*

Edad: *28*

Ciudadanía: *Cubana*

Institución: *Universidad de las Ciencias Informáticas (UCI)*

Título: *Ingeniero Informático*

Categoría Docente: *Profesor Instructor*

E-mail: *alexanderfs@uci.cu*

Ocupación Actual: *Jefe del grupo de Arquitectura y Tecnologías, Dirección Técnica, IP – UCI*

AGRADECIMIENTOS

En primer lugar le doy gracias a mi Dios, Señor y Salvador Jesús el Cristo que ha sido el principal causante de que hasta aquí haya llegado. Él ha estado dándome fuerzas y ánimo cada día, haciéndome entender que está y estará conmigo siempre a pesar de cualquier cosa que esté pasando a mi alrededor.

Le doy gracias a mis compañeros de grupo que me han soportado y compartido conmigo estos 5 años de universidad.

Agradezco a todos aquellos profesores y personas que por cuestiones de la vida nos hemos distanciado, pero que influyeron en mí para formar mi carácter y mi educación especialmente a: Valdi, Marisel, Carlos y su esposa Juana, a Oliver por enseñarme a programar y a pensar.

A Mileivys por prestarme su oído incondicionalmente, gracias a Brígida por confiar en mí.

A mis tutores Eduardo Lago e Igr por enseñarme que la vida es dura, pero que siempre hay solución y siempre las mejores son las más sencillas.

A mi familia por mantenerme vivo.

También quiero agradecer de una manera especial a todos los cristianos que conozco en la universidad, a mis hermanos en Cristo, que me han ayudado a entender que la vida es una universidad mucho más larga y difícil pero que con Cristo es menos monótona y más amena.

Alberto

AGRADECIMIENTOS

Quisiera agradecer ante todo a mis padres, en especial a mi madre, por ser todo para mí, por asumir mi educación con tanta destreza y valentía, por guiarme siempre por el camino correcto. A mi hermano, por apoyarme en todas mis decisiones, por contribuir a mi formación como profesional, por ser mi padre, cuando tuvo que serlo.

A mi tío Orestes, el "general" más grande que he conocido; que cuando llegué a esta agitada ciudad, me estrechó en sus brazos casi sin conocerme y me dio el apoyo que necesitaba. Gracias por enseñarme que la comida sana también llena, que las personas no siempre hacen las cosas que uno espera y que siempre es mejor tomar precauciones. Gracias por ser un padre de verdad.

A mis primos, por su constancia, en especial a Yasser por ser mi hermano incondicional. Gracias por elevar el concepto de familia a su más alta expresión.

A Rocy y a Abel, por ser mis otros hermanos más que mis cuñados, por su sabiduría, su ejemplo y sus frijoles. A Raiza, por soportarme todos esto años, por su infinito e inagotable amor.

A mis abuelos, en especial a mi abuelito Lombera que tanto deseó verme en este momento, pero que al final no pudo. Esto es para ti también, con toda la fuerza del mundo, donde quiera que estés.

A todos los profesores de la UCI que tuvieron que ver con mi formación, en especial a mis tutores Lago e Igr por su tesón, su confianza, sus sugerencias, sus conocimientos. Gracias por enseñarnos que la vida, además de todo, es dura, pero que siempre hay soluciones y a menudo la mejor es la más sencilla.

A todos mis amigos, en especial a los integrantes del proyecto "Juegos Consola", compañeros de lucha científica, de arduo batallar por ser cada día mejor. Gracias a todos por permitir que este sueño se haya materializado.

Lombera

DEDICATORIA

A mi madre, hacedora de virtudes, faro que guía cada uno de mis pasos. La mujer más bella que jamás he conocido. Atribuyo todo lo que soy, todos mis éxitos en esta vida, a la enseñanza moral, intelectual y física que recibí de ti.

Lombera

RESUMEN

La presente investigación propone primeramente una novedosa técnica de animación, los sistemas articulados de cuerpos rígidos y luego un entorno de modelación tridimensional para su eficiente construcción, edición y prueba. Dicho entorno consta de disímiles facilidades que permiten abstraer a los desarrolladores, de los modelos matemáticos de integración subyacente para crear dichas animaciones, así como de complejas rutinas de simulación, permitiendo de esta manera a la imaginación jugar un papel primario. Además, queda evidenciada la potencia de bibliotecas gráficas y motores físicos como *Graphics Three Dimensional Engine* y *Open Dynamics Engine*, capaces de acelerar el proceso de desarrollo a corto plazo en un mercado tan competitivo donde la palabra reutilización se impone. Se describen también los entornos de modelación tridimensional existentes más utilizados hoy en día, los cuales sirvieron de inspiración a los creadores; las técnicas, herramientas, metodología utilizada y un modelo unificado de eventos que por su poder de abstracción y facilidad de uso está llamado a suplir las necesidades de los programadores de C++, relacionadas con la manipulación de los mismos. Por último, la descripción de un modelo de persistencia con ayuda de un sencillo *framework* para este propósito, garantiza la reutilización por terceras personas de los ficheros generados por el *software*, minimizando de esta manera los recursos para la creación de los sistemas articulados de cuerpos rígidos e impactando positivamente en la eficiencia de los desarrolladores. Se provee además la descripción arquitectónica de la solución propuesta.

Palabras Clave:

Animaciones, sistemas articulados de cuerpos rígidos, delegados, persistencia, modelación.

ÍNDICE

ÍNDICE

ÍNDICE

INTRODUCCIÓN

Una animación es una producción de imágenes consecutivas que cuando son desplegadas transmiten una sensación de movimiento. Las animaciones son casi mágicas por sus habilidades para capturar nuestra imaginación. Maravillándonos con efectos especiales o hipnotizándonos con movimientos puros, las animaciones pueden infundir a una secuencia de imágenes inertes la ilusión de la vida y el movimiento. Crear esta ilusión manualmente o asistido por *software* no es una tarea trivial. Esto se debe, entre otras razones a que cada imagen individual o cuadro, en la secuencia, debe mezclarse suavemente con las imágenes consecutivas, con la finalidad de crear un movimiento continuo a través del tiempo.

La animación por computadora puede ser generada en tiempo real para usarse en videojuegos y otros medios interactivos. Así, es posible que un humano controle un personaje interactivo en el desempeño de sus acciones. Las técnicas reales de animación y *rendering* posibilitan la creación de actores digitales que pueden ser suavemente mezclados con escenas del mundo real. Estas técnicas se pueden agrupar en dos clases principales: bidimensionales (2D) y tridimensionales (3D), constituyendo las últimas el centro de atención del presente trabajo.

Las animaciones 3D incluyen la construcción de un mundo virtual en el cual los caracteres y los objetos se mueven e interactúan. El programador debe modelar, animar y renderear la escena 3D. Así, la modelación incluye la descripción de los elementos de la escena y su colocación apropiada; la animación, por otra parte, se refiere cómo los objetos deben moverse en dicha escena. Para animar un cuerpo se necesita la descripción estática del objeto y la información adicional de cómo este se mueve. Una forma común de especificar esta información es usando un modelo articulado. Estos permiten coleccionar objetos conectados entre sí por articulaciones en una estructura jerárquica, en la cual la localización de un objeto está determinada por la localización de su objeto padre en dicha jerarquía. Por ejemplo, el movimiento de la articulación del codo en el modelo de un humano afectará no solamente la posición del antebrazo sino también la posición de la mano y los dedos. El objeto padre en la jerarquía, puede ser movido arbitrariamente para controlar la posición y la orientación del modelo completo.

Un segundo tipo de modelo utilizado en la animación son los sistemas de partículas, en los cuales el movimiento de las partículas a través del espacio está determinado por un conjunto de leyes de la Física, de manera que estas caen por gravedad y colisionan con otros objetos en el entorno. Dentro de los

sistemas propensos a ser modelados por sistemas de partículas se incluyen: el humo, la lluvia, las explosiones y los bandos de pájaros. Incluso los cuerpos rígidos, para los cuales se ignoran las deformaciones, pueden ser simulados a través de sistemas de partículas (1).

Los objetos deformables constituyen otro tipo de modelo utilizado en la animación por computadora. Existen diferentes formas para representar objetos deformables dentro de las cuales se incluyen: los sistemas de resortes, los modelos volumétricos, los sistemas de partículas vibratorios y la representación de superficies. El agua, el pelo, la ropa y los peces están dentro de las entidades reales que pueden ser simuladas satisfactoriamente como objetos deformables. Mientras que cada uno de estos tipos de modelos puede ser usado para describir una amplia variedad de objetos, los sistemas complejos requieren con frecuencia modelos híbridos que combinen dos o más tipos. Precisamente, esta estrategia será la utilizada en la presente investigación en la cual conceptos como los modelos articulados, los cuerpos rígidos y los sistemas de partículas podrán ser extendidos a sistemas articulados de cuerpos rígidos, que permitirán la representación de distintas animaciones independientemente del modelo de integración subyacente que se use y de toda la implicación matemática que esto conlleva, garantizando así un nivel de realismo adecuado. En un sistema articulado de cuerpos rígidos, los objetos se encuentran atados mutuamente usando varias conexiones, las cuales pueden ser representadas y personalizadas como objetos y se diferencian en los grados de libertad asociados a su movimiento (2).

Por otra parte, el *hardware* de la computación gráfica está cambiando muy rápidamente, de manera que las API (del inglés, *Application Program Interface*) son necesarias para proveer una interfaz consistente y portable para las implementaciones de *hardware* de cualquier funcionalidad. Así, al encapsular una funcionalidad los desarrolladores no necesitan implementarla repetidamente. De esta forma, la aplicación pueda ser fácilmente portada a cualquier otra plataforma que posea dicha API.

Situación Problémica

El proyecto “Juegos Consola” de la Facultad 5 perteneciente a la Universidad de las Ciencias Informáticas (UCI), especializado en el desarrollo de juegos virtuales, actualmente se encuentra enfrascado en la confección de juegos de autos. Para la creación de los modelos físicos asociados a los modelos visuales, los programadores deben crear un sistema articulado de cuerpos rígidos componiendo primitivas básicas (cajas, cilindros, esferas, cápsulas) hasta que coincida o se aproxime a su descripción visual, para dotarlo así de realismo y facilidad al manipular las colisiones. El modelo requerido, tiene el propósito de establecer varias funcionalidades tales como: la dirección del auto, la suspensión, las puertas, aprovechando al máximo las funcionalidades de la biblioteca física utilizada actualmente para la representación de sistemas articulados. Una vez creado este, resta solamente que se definan las propiedades de las distintas primitivas en las que se divide, situando correctamente cada una y estableciendo sus dimensiones. Dicho procedimiento resulta costoso para los desarrolladores de los modelos físicos quienes tienen que definir mediante instrucciones de código cada uno de los mismos, resultando sin dudas una operación tediosa y abrumadora para los modelos complejos.

La ineficiencia actual de los procesos de diseño, edición y prueba de sistemas articulados de cuerpos rígidos, durante la confección de videojuegos en el proyecto “Juegos Consola”, atenta contra el buen desempeño de los desarrolladores de los modelos físicos. Por esta razón, se impone la búsqueda de una solución para esta problemática.

Problema Científico

¿Cómo realizar eficientemente los procesos de diseño, edición y prueba de sistemas articulados de cuerpos rígidos durante la construcción de videojuegos en el proyecto “Juegos Consola”?

Objeto de Estudio

Las técnicas de animación computacional en el desarrollo de videojuegos.

Campo de Acción

Los sistemas articulados de cuerpos rígidos en el desarrollo de videojuegos.

Objetivo General

Elaborar una herramienta para la edición de sistemas articulados de cuerpos rígidos.

Objetivos Específicos

1. Diseñar el entorno de modelación 3D con una Interfaz Gráfica de Usuario amigable.
2. Crear y editar sistemas articulados de cuerpos rígidos en el entorno de modelación 3D.
3. Exportar los sistemas articulados a un fichero e importarlos o a partir de este, utilizando el entorno de modelación 3D.
4. Probar los sistemas articulados de cuerpos rígidos en el entorno de modelación 3D.

Hipótesis

La creación y utilización de los sistemas articulados de cuerpos rígidos aumentará de manera significativa el nivel de realismo y la eficiencia durante el desarrollo de los videojuegos en el proyecto “Juegos Consola”.

Variables

Independiente:

- Sistemas articulados de cuerpos rígidos

Dependiente:

- Nivel de realismo
- Eficiencia

Métodos Científicos

Empíricos

Observación

La observación como percepción planificada, relativamente prolongada, de un hecho o fenómeno estrechamente relacionado con los sistemas articulados, permitió corregir o reafirmar un modelo escogido.

Consulta a expertos

Las consultas a diferentes expertos en el tema permitió la adquisición de nuevos conceptos y experiencias para la investigación, muchas veces no documentada.

Teóricos

Análisis – Síntesis

Este método permitió la descomposición de los sistemas articulados en sus múltiples relaciones y componentes para facilitar su estudio. A la vez, estableció las pautas para la unión de las partes previamente analizadas, posibilitando así descubrir sus características generales y las relaciones esenciales entre ellas.

Hipotético – Deductivo

A partir de la hipótesis y siguiendo reglas lógicas de deducción se arribó a nuevos conocimientos y predicciones, los que posteriormente fueron sometidas a verificaciones empíricas.

Modelación

Mediante este método se crearon abstracciones con el objetivo de explicar la realidad. Se tuvo en cuenta en todo momento la correspondencia objetiva entre el modelo y el objeto.

Tareas de la Investigación

1. Analizar los logros y limitaciones en los enfoques existentes sobre sistemas articulados de cuerpos rígidos.
2. Evaluar el contenido de la información obtenida sobre sistemas articulados de cuerpos rígidos.
3. Establecer un diagnóstico de las tendencias actuales y tomar posición al respecto.
4. Poner en marcha la biblioteca gráfica *Graphics Three Dimensional Engine* en su versión 7.00 y la biblioteca física *Open Dynamics Engine* en su versión 0.9.
5. Analizar las bases teóricas de la biblioteca física *Open Dynamics Engine*.
6. Establecer el prototipo del entorno de modelación 3D.
7. Diseñar e implementar una arquitectura de clases que de soporte a las funcionalidades de creación y edición de sistemas articulados de cuerpos rígidos.

8. Establecer el formato del fichero que utilizará el editor para exportar e importar el sistema articulado de cuerpos rígidos creado.
9. Diseñar e implementar un entorno virtual de prueba para el sistema articulado de cuerpos rígidos creado.
10. Diseñar e implementar un *framework* para la simulación física basado en la biblioteca *Open Dynamics Engine*.

Con la realización de este trabajo se pretende proveer a la comunidad de desarrolladores de videojuegos de la UCI, de una herramienta capaz de perfeccionar las respuestas ante diferentes eventos de un sistema articulado de cuerpos rígidos. La misma permitiría exportar los ficheros creados y su utilización por terceros, minimizando de esta manera los recursos para la creación de los mismos e incrementando la productividad de los desarrolladores.

El trabajo está estructurado en cuatro capítulos. En el Capítulo 1 denominado *Fundamentación Teórica,* se introducen todo los conceptos necesarios para la comprensión del presente trabajo y se recogen los antecedentes del tema tratado a nivel internacional, nacional y de la UCI. Se incluye además un estudio de las tendencias, técnicas, tecnologías, metodologías y *software* usados en la actualidad, en los que se apoya la solución propuesta. En el Capítulo 2 titulado *Características del Sistema* se brindará información detallada del objeto de automatización, describiéndose además el modelo del negocio. En una segunda etapa se especificarán los requisitos de *software*, se definirán los actores del sistema así como el diagrama de casos de uso del mismo. Posteriormente, se analizará un modelo unificado de eventos propuesto para el lenguaje C++ sobre el cual descansa la aplicación desarrollada. Como último punto, se describirá el modelo de persistencia utilizado para el almacenamiento del modelo físico creado en el entorno de modelación. En el Capítulo 3 nombrado *Análisis y Diseño del Sistema*, utilizando el Lenguaje Unificado de Modelado (UML) quedarán plasmados los diagramas de clases del análisis, de clases del diseño y de interacción, como artefactos fundamentales en esta fase. Los mismos han sido separados por casos de uso para facilitar su comprensión. Se describirá también el tratamiento de errores, una concepción general de la ayuda del sistema así como los patrones de diseño utilizados. Por su parte, el Capítulo 4 titulado *Implementación* expondrá el artefacto Modelo de Implementación, que incluye la especificación de cada uno de sus subsistemas, destacándose al final las reglas de codificación que se siguieron durante la realización de dicho flujo de trabajo.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

A la vez que el campo de la Computación Gráfica se hace más sofisticado, se demanda más nivel de realismo en sus escenas y por tanto, el grado de excelencia y las interacciones se incrementan al hablar de los efectos generados computacionalmente. Los modelos articulados y los sistemas de partículas han sido usados en la animación por computadora por muchos años con este propósito y más recientemente en la simulación en tiempo real y en los videojuegos. Los primeros han servido para especificar la postura de los caracteres así como sus movimientos, mientras que con los segundos ha sido posible simular una amplia gama de fenómenos físicos de manera muy fácil, sin que para ello se necesiten grandes cantidades de memoria. Mucho más potente ha sido su combinación, la cual ha permitido hablar de sistemas articulados de cuerpos rígidos, capaces de representar sistemas complejos con un alto nivel de realismo.

A lo largo del presente capítulo se expondrá la historia de cada una de las técnicas mencionadas anteriormente así como los responsables de sus descubrimientos. Se hará referencia al entorno de desarrollo escogido para realizar la implementación de la herramienta, abundando sobre sus características. Al mismo tiempo, abordaremos las tendencias actuales relacionadas con la construcción de editores gráficos, cuestión esta que permitirá tomar posición al respecto para la implementación del mismo. Posteriormente, se hará referencia a las bibliotecas gráficas disponibles en la actualidad, capaces de satisfacer las necesidades de desarrollo. Se resaltará sus resultados y potencialidades, concretando de esta manera la elección de la más adecuada. De igual forma se analizarán los motores físicos o bibliotecas físicas de alto rendimiento disponibles, pues constituyen el núcleo para la simulación de estos sistemas.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

1.1 Sistemas de Partículas

El método para la modelación de fenómenos naturales y objetos que exhiben características similares a la de los fluidos o propiedades caóticas es llamado *sistema de partículas*. Este método es particularmente bueno para la descripción de objetos que cambian o se mueven en el tiempo. Dentro de estos objetos podemos citar: el agua, el polvo, las nubes, las manadas de animales y las explosiones.

El concepto sistema de partículas no es una idea nueva. Los mismos fueron por primera vez usados en los juegos primitivos de *Atari* para modelar las explosiones y los disparos. Reeves en su artículo de 1983 trajo los sistemas de partículas a la luz de la computación gráfica cuando los utilizó para crear el efecto de la bomba Génesis en la película Viaje a las Estrellas II (3). Su especificación de qué atributos tenía una partícula y de cómo un sistema de partículas era calculado prevalece en las más recientes investigaciones. Las partículas de Reeves eran de forma esférica, rectangular o circular. Su valor inicial de velocidad estaba atado firmemente a su posición inicial, mientras los otros atributos para las partículas tales como el color eran generados estocásticamente. Por otra parte, la dinámica de los sistemas de partículas originales era muy simple. La única fuerza que afectaba a una partícula después de su creación era la gravedad, retornando trayectorias parabólicas para todas las partículas. De esta forma, la gran complejidad alcanzada por estos sistemas giraba alrededor de los valores iniciales generados estocásticamente y no por la dinámica.

El siguiente progreso en los sistemas de partículas una vez más vino de la mente de Reeves y se publicó en su artículo titulado *"Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems"* (4). Había creado una forma más compleja de renderear sistemas de partículas a la vez que hacía nuevas estructuras que podían producir más efectos y objetos complejos tales como hierbas y árboles. Precisamente, el cambio más notable respecto a su primer sistema fue el proceso estocástico de modelación que empleó para el *rendering*. Las partículas, esta vez conformando árboles, no eran emisoras de luz como en su primer artículo; por el contrario debían reflejarlas, así que un nuevo sistema para renderear estos nuevos objetos tenía que ser implementado. Dado que el *rendering* individual de cada rama u hoja sería una pesadilla computacional, algunas suposiciones estocásticas tuvieron que hacerse en pos de salvar el tiempo de *rendering*. El segundo aporte de este artículo fue que le dio al

sistema de partícula una complejidad superior que permitió la creación de objetos más complejos. Una implementación de este sistema puede ser visto en el animado “Las aventuras de André y Wally B” (5).

Hasta su investigación, la computación gráfica se componía principalmente por figuras creadas a partir de polígonos y conjuntos de vértices. La investigación de Reeves permitió la definición de objetos cuyas aristas estuvieran poco definidas. Este nuevo método permitió la creación de efectos tales como: la nieve, la lluvia, el fuego y las nubes. El agua es un efecto particular que se adecúa bien a los sistemas de partículas. Peachey en 1986, usó sistemas de partículas para modelar el rociado de las olas rompientes y del agua cuando colisiona con objetos (6). Alrededor del mismo tiempo Fournier utilizaba sistemas de partículas de la misma manera para modelar la espuma producida por las olas del océano (7).

Otro uso para los sistemas de partículas fue descubierto por Reynolds en su artículo de 1987 *“Flocks, Herds and Schools: A Distributed Behavioral Model”* en el cual utilizaba sistemas de partículas para modelar un bando de aves las cuales viajaban juntas alrededor de la pantalla (8). Esta vez, el sistema de partículas interactuaba entre sí y cada partícula en lugar de ser solamente puntos o figuras simples eran objetos que tenían una forma geométrica, comportamiento y orientación. A estas partículas les llamó *Boids*[1]. Los mismos podían ser vistos como sistemas de partículas extendidos adicionándoles un sistema de coordenadas a cada partícula, modelando así un movimiento más complejo para cada una especialmente los movimientos resultantes de la interacción con otra partícula. Directamente se adopta esta técnica en la presente investigación para representar cada uno de los movimientos, garantizando así un alto nivel de realismo. Tiempo después, Lisa Schaefer expandió el trabajo de Reynolds al añadirle simples reglas de si/no al sistema, representando grupos de personas que vagaban por las calles para que pudieran interactuar con el tránsito que se avecinaba (9).

El humo y el polvo han sido también objeto de muchas investigaciones. Chen en el año 1999 usó un modelo físico empírico para crear y controlar partículas de polvo provenientes de un vehículo en movimiento. Separó las partículas de polvo en tres escenas y estableció un simplificado sistema de partícula para cada una. Esta alternativa aceleró los cálculos computacionales y permitió al sistema ser rendereado en la pantalla en tiempo real. El sistema creado por Chen era estable y no sufría de disipación

[1] Boids: abreviatura de Birdoid

numérica. Además, tenía un gran desempeño en un momento donde los detalles visuales y la velocidad eran cruciales (10).

Muchos otros usos de los sistemas de partículas han sido encontrados por otros investigadores. Por ejemplo, Kajira utilizó en 1989 la idea de los *texel*[2] para renderear el pelaje de los animales. Sus *texels* fueron básicamente una forma de renderear partículas en un sistema *ray-traced* (11). Otras investigaciones como la de Szeliski usaron un sistema de partículas orientado para modelar superficies las cuales podían ser deformadas, cortadas y manipuladas (12). De la misma manera Crossno creó *isosurfaces*[3] con sistemas de partículas, que poseían partículas que se atraían o repelían unas a otras. Cuando el sistema alcanzaba el equilibrio las posiciones de las partículas eran usadas como un mapa, para con polígonos crear la superficie (13).

Un año después, en 1993, Leech describió un entorno con un conjunto de herramientas de desarrollo para un sistema de partículas virtual interactivo. Usando un casco virtual el usuario tiene acceso a un conjunto de controles donde cada uno representa una operación del sistema de partícula. Los controles pueden ser interactivamente ajustados y colocados en el entorno para ver los resultados de los efectos especiales de los sistemas de partículas. El principal objetivo del artículo era la inmersión en el sistema de interacción gráfica (14). Por su parte Dorsey utilizó un sistema de partículas para modelar el deterioro de superficies así como el paso del agua a lo largo de una superficie. Sus métodos fueron muy provechosos en la modelación de rocas y estructuras a la intemperie (15).

Los sistemas de partículas se han usado extensivamente en la industria de los juegos, pues pueden crear impresionantes efectos visuales con poco tiempo de procesamiento. Sims en su artículo de 1990 esboza muchos de los usos más comunes para sistemas de partículas y propone una forma de usar computación en paralelo para renderear las partículas. Su artículo constituye un buen escalón para el diseño rápido y eficiente de técnicas de modelación para sistema de partículas (16).

Los sistemas de partículas han sido cada vez más investigados en los últimos años, a partir de que las computadoras incrementan su velocidad. Esto ha permitido la realización de muchos efectos que no eran

[2] Texel: en Inglés abreviatura de elemento de textura (texture element).

[3] Isosurface: superficie de nivel.

factibles en años anteriores debido al costo de procesamiento. Foster en su artículo *"Practical Animation of Liquids"* desarrolló un modelo general basado en sistemas de partículas para la modelación y animación de fluidos específicamente diseñado para la animación por computadora. Este sistema ha sido empleado en numerosas películas de *Hollywood* para manipular líquidos viscosos que interactúan con entornos 3D (17).

Tal y como se mencionaba anteriormente los videojuegos han comenzado a usar los sistemas de partículas en tiempo real para efectos especiales. *Quake III Arena* tiene partículas de chispa y partículas de humo. Estos sistemas de partículas parecen contener entre cien y quinientas partículas y son rendereadas como puntos o polígonos alineados en la pantalla con texturas. Su comportamiento se rige principalmente por la gravedad y un color pálido, con complejas velocidades iniciales, tales como espirales. Un paquete más comercial de animación como el Maya y el 3D Studio Max incluyen ahora un paquete de animación de sistemas de partículas. Referente a las API se puede decir que existe un antecedente en David K. McAllister con su artículo *"The Design of an API for Particle Systems"* donde se propone una biblioteca realizada en C++ que le permite a las aplicaciones simular la dinámica de partículas en la creación de efectos especiales (18).

1.2 Modelos Articulados

Los modelos articulados constituyen colecciones de objetos conectados entre sí por articulaciones, en una estructura jerárquica semejante a un árbol. La génesis de estos modelos para la representación de articulaciones puede ser encontrada en estudios de cinemática relacionados con los manipuladores robóticos. Estos sistemas usaban la notación de los parámetros de *Denavit-Hartenberg* de la robótica para representar maniquíes virtuales con extremidades articuladas (19). Aunque la notación para asociar los sistemas de coordenadas entre segmentos adyacentes utilizado por esta técnica es conveniente, cada parámetro ajustado describe solamente un solo grado de libertad entre dos segmentos. Múltiples grados de libertad pueden ser entonces alcanzados combinando múltiples ajustes de estos parámetros.

Así, estructuras como la columna vertebral que exhiben un alto grado de comportamiento acoplado han sido explotadas por investigadores como Monheit para desarrollar un modelo cinemático de una columna

que exhibía movimientos de flexión-extensión, encorvamiento lateral y una rotación y torsión axial (20). Para la representación normalizada de un esqueleto humano Kulpa modeló una columna con un spline que podía ser dividido en segmentos. Para recuperar las posiciones de las vertebras, el spline que representaba la columna era simplemente discretizado teniendo en cuenta las distancias de las vertebras (21). Dejando aparte a la propia columna, Maciel incorporó articulaciones que podían trasladarse y rotar al mismo tiempo sobre un plano; vale destacar que los límites de las articulaciones cambiaban dinámicamente con los grados de libertad de cualquier otra (22). Más de cerca Herda caracterizó el comportamiento acoplado de las articulaciones a través de superficies implícitas obtenidas a partir de los valores recuperados en una captura de movimiento (23). Esta representación permitió caracterizar las dependencias de las articulaciones, pero es una forma no muy intuitiva para controlar los movimientos de los humanos (24).

Para restringir el movimiento de una articulación de rótula *(ball & socket)* la técnica de los polígonos esféricos y la de los conos huecos fueron introducidas (25) (26). Los polígonos esféricos son más generales que los conos pero son más complejos de tratar. Los conos son suficientes para representar articulaciones humanas (25).

Shao en el año 2003 introdujo un componente general para articulaciones llamado mapa de articulaciones que permitía modelar expresiones de articulaciones bajo numerosos segmentos de huesos, ideal para precisas articulaciones biomecánicas (27). Un mapa de articulaciones es una función que toma un conjunto de entradas (e.g. conjunto de articulaciones) para producir como salida una o más articulaciones. La entrada puede ser vista como los grados de libertad y la salida son los valores de la articulación modificada (e.g. ángulos, traslación, variación). La salida de un mapa de articulaciones puede ser combinada con las salidas de otro mapa para crear comportamientos cada vez más sofisticados. El proceso que ejecuta este mapeado varía según el tipo de comportamiento deseado para la articulación.

Todas estos métodos originados a partir de la mecánica y la robótica han sido adaptadas por muchos investigadores a simulaciones en otras ramas estrechamente ligadas a la animación por computadora, específicamente la simulación de sistemas articulados de cuerpos rígidos, tema que nos toca de cerca y en el cual cada cuerpo se encuentra enlazado a otro a través de una articulación. Existen un sinnúmero de

tipos de articulaciones con las que pueden ser conectadas estos cuerpos, las cuales difieren unas de otra por los grados de libertad asociados a su movimiento. Murilo hizo toda una disertación en su libro *"Dynamic Simulation of Multibody System"*. En su capítulo 5 describe los principales métodos para guiar la dinámica de los sistemas articulados, así como sus dos clasificaciones que incluyen la descripción del movimiento completo del sistema articulado garantizando las restricciones de las articulaciones (2). Este método es conocido como la Reformulación Lagrangiana basado en los Multiplicadores de Lagrange. Otro afamado libro de la rama, *"Computational Dynamics"* de Ahmed A. Shabana realiza una introducción a los conceptos, definiciones y técnicas utilizadas en la dinámica de sistemas articulados cubriendo interesantes tópicos relacionados con el movimiento en tres dimensiones (28).

Jihun Park en 1997 presentó una nueva metodología para modelar y controlar el movimiento de un cuerpo rígido articulado (29). La técnica usaba un método de optimización de parámetros en la simulación dinámica para obtener un buen conjunto de valores que le permitiera controlar las variables del sistema (fuerzas y torques). Dicho sistema articulado era modelado usando nodos de control.

En el año 2007 Jan Bender durante su tesis de doctorado desarrolló un nuevo método para la simulación de sistemas articulados de cuerpos rígidos, "La Simulación Dinámica basada en Impulsos" (30). El Dr. Bender asegura que su método basado en impulsos tiene algunas ventajas en relación con los métodos clásicos como el de los Multiplicadores de Lagrange, asevera además que es fácil de usar y que puede manipular todos los tipos de articulaciones, entre otras facilidades. Sin dudas una tentadora oferta, que bien vale para otra investigación.

1.3 *Software* Integrado de Modelación 3D

El producto a realizar debe contar con un editor de sistemas articulados. Es decir, debe contar con una interfaz gráfica que permita al usuario diseñar modelos 3D. En la actualidad existen muchísimos *software* que se dedican a la modelación 3D tales como: 3D Studio Max, Maya, Blender por solo citar algunos. Estos permiten al usuario crear objetos y ambientes virtuales de formas variadas y probar su interacción. Todo este proceso se logra de una manera muy cómoda, eficiente y con calidad. Sin embargo, todos estos *software* se utilizan para crear videos y multimedias generalmente; es decir, no permiten de manera

sencilla la inmersión de estos mundos virtuales en un entorno interactivo. Esta característica unida a que son costosos y que realmente se necesita una pequeña parte de las potencialidades de los mismos (la animación de sistemas articulados), los descarta como herramientas a utilizar en la resolución del objetivo del presente trabajo.

Es por esto que el futuro producto contará con su propia Herramienta Integrada de Modelación 3D que permitirá resolver el problema específico. Además posibilitará exportar el diseño hecho en esta, para que pueda ser insertado en un entorno virtual interactivo.

A continuación una lista de los *software* actuales más utilizados en la modelación 3D y un resumen de sus características.

1.3.1 3D Studio Max

3D Studio Max es usado en muchas industrias que utilizan gráficos 3D como es el caso de la cinematografía. Es usado también en la industria de los juegos para crear modelos gráficos. Es uno de los más costosos de su tipo ($3500 USD) y corre en Windows. Actualmente está en su versión “2008” Versión 10. Este producto presenta una cómoda interfaz de usuario y se toma como base para la creación de la interfaz gráfica de la aplicación (ver figura 1).

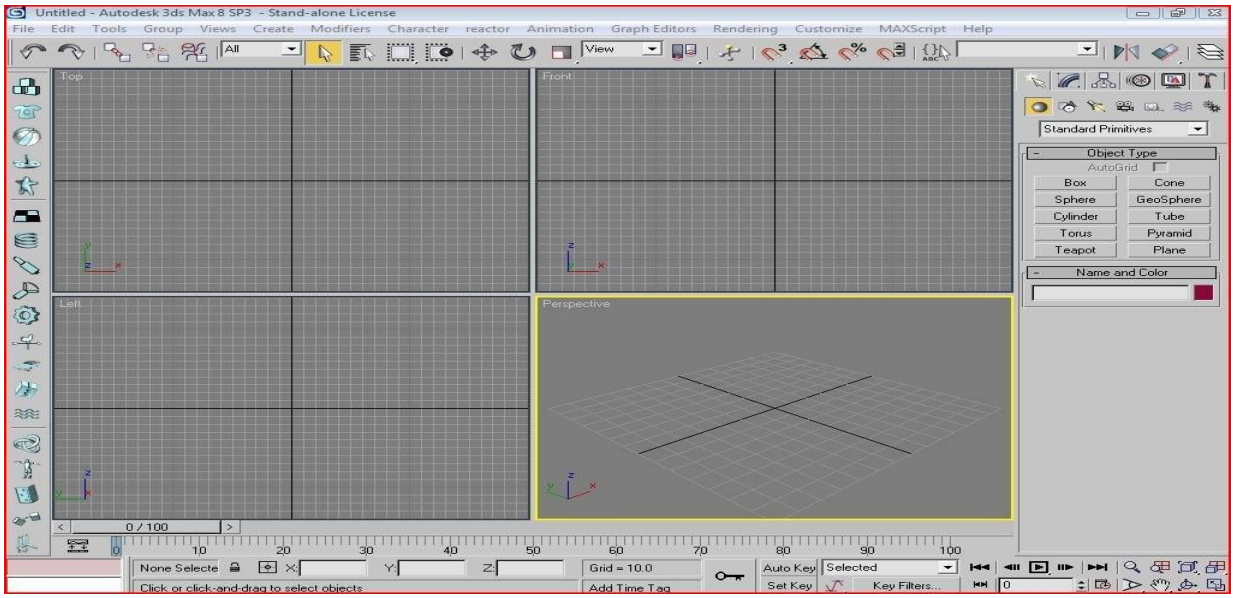


Figura 1: Interfaz del 3D Studio Max.

1.3.2 Cinema4D

Es una herramienta que presenta menos potencialidades que otras de su tipo pero está pensada para especializarse en el diseño de piezas. Corre sobre MAC OS X, Windows y Linux. Actualmente está en la versión 10.5.

1.3.3 Maya

Es actualmente utilizada en la industria cinematográfica y presenta una curva de aprendizaje bastante alta. Esta herramienta permite, al igual que 3D Studio Max, la animación física aunque es usada mucho más por sus gráficos. Actualmente está en su versión “2008” Versión 9 y su costo es de $2000 USD en la versión Maya Complete y $7000 USD en la versión Maya Unlimited.

1.3.4 Blender

Es una herramienta libre y de código abierto. Es usada para animaciones, modelado y presenta características de texturizado, *rendering* y animaciones tan buenas como Maya, 3D Studio Max o

Cinema4D. Además de sus excelentes características como el Modelado 3D, permite la animación de sistemas de partículas y simulación de fluidos. Tiene licencia GPL y corre en muchos sistemas operativos como Windows, OS X, Linux, Free BSD, Sun e Irix.

1.3.5 Modo

Es utilizada específicamente para modelación y el *rendering*. Recientemente se le ha agregado un modulo de animación. Su costo es de $900 USD y corre en Windows y MAC OS X. Actualmente se encuentra en su versión 3.01.

1.3.6 ZBrush

Dicha herramienta es utilizada solo para modelar, su costo es de $489 USD. Actualmente está en su versión 3.1.

Podemos concluir que todas estas herramientas son utilizadas potentemente para la modelación 3D y que algunas incluyen un módulo de animación y simulación. No obstante todas son costosas y además su objetivo principal como se dijo anteriormente no es la simulación física sino la modelación. Por todo esto se hace necesaria la construcción de una herramienta para crear simulaciones y probarlas, con una sencilla interfaz gráfica para la modelación 3D, la cual estaría basada en el prototipo de interfaz de 3D Studio Max (por su comodidad), definiéndole solamente las funcionalidades requeridas por la aplicación.

1.4 Motor Gráfico

En computación el término *game engine* hace referencia al componente principal, al alma de un videojuego. Típicamente manipula el *rendering* y otras tecnologías necesarias, pero pudiera también manipular cuestiones adicionales como la Inteligencia Artificial y la Detección de Colisiones entre objetos. Los elementos más comunes que un *game engine* provee son las utilidades gráficas de *rendering* ya sean 2D o 3D. Los *engines* que solamente proveen *rendering* 3D en tiempo real son con frecuencia llamados 3D *engines* o motores gráficos. Existen un sinnúmero de ellos, alrededor de 284 (registrados), y cada uno con sus propias potencialidades que los describen auténticamente.

Entre los motores gráficos comerciales y *open source*[4] más utilizados por la comunidad gráfica internacional podemos citar:

Comerciales	***Open source***
Torque Game Engine	*Object Oriented Graphics Rendering Engine*
TV3D SDK 6.5	*Graphics Three Dimensional Engine*
3DGameStudio	*Irrlicht*
C4 Engine	*Crystal Space*
Unity	*Panda3D*
NeoAxis Engine	*JME*
DX Studio v2.2	*Reality Factory*
3Impact	*The Nebula Device 2*
Beyond Virtual	*RealmForge*
Deep Creator	*Blender Game Engine*

Tabla 1. Prestigiosos motores gráficos en la comunidad gráfica internacional.

A continuación se analizarán las principales características de dos de los motores gráficos más prestigiosos en la comunidad.

1.4.1 Graphics Three Dimensional Engine

G3D *Engine* (del inglés, *Graphics Three Dimensional Engine*) es un motor gráfico escrito en C++, de calidad comercial, *open source* bajo la licencia BSD (31). Ha sido usado en juegos comerciales, artículos investigativos, simuladores militares y en las universidades. Soporta *rendering* en tiempo real, *rendering* desconectado (*off-line rendering*) y cálculos de propósito general basado en los GPU (del inglés, *Graphics Processing Unit*). G3D facilita un conjunto de rutinas y estructuras comunes que son necesarias en casi todos los programas de gráficos. Hace las bibliotecas de bajo nivel como OpenGL (del inglés, *Open Graphics Library*) y los *sockets* más fáciles de usar y sin limitar su funcionalidad o rendimiento. G3D es sólido como una roca, con una base altamente optimizada sobre la cual pueden construirse aplicaciones 3D. Su equipo de desarrollo se extiende a toda la industria gráfica. Incluye desarrolladores de juegos profesionales, militares y contratistas, doctores, estudiantes y profesores. Algunas características en G3D

[4] Open source: dícese de los programas cuyo código fuente original esta disponible, sin restricciones.

son útiles para cualquier programa, sin importar si este ejecuta cálculos 3D o corre sobre un procesador gráfico. Cuando se usa G3D como una biblioteca utilitaria no es necesario el *framework* para la GUI (del inglés, *Graphical User Interface*). Para soportar usos utilitarios la biblioteca se divide en dos partes: G3D.lib y GLG3D.lib. Toda la manipulación de eventos, de ventanas y código OpenGL está en la porción GLG3D.lib. Esto significa que se puede usar G3D.lib sin estar atado a OpenGL o al establecimiento de alguna rutina de manipulación de eventos. Su versión 7.00 constituye una completa solución gráfica para la construcción de juegos 3D y simuladores. Contiene poderosas características como una GUI parametrizable, soporta la carga de numerosos formatos de modelos 3D, así como eficientes mecanismos para el trabajo con *shaders*. Es utilizada en muchas universidades prestigiosas, incluyendo la *Universidad de Brown (E.U)*, *el Colegio Williams (E.U)*, *la Universidad de Ulm (Alemania), la Universidad de Kent (Inglaterra), Harvard Extension School (E.U), la Universidad de Carolina del Norte (E.U), la Universidad Guang Dong (China), la Universidad del Estado de Georgia (E.U), y el Instituto Politécnico Rensselaer (E.U)*, apareciendo además en numerosos juegos comerciales.

A continuación, algunas de sus características:

Autor	Morgan McGuire
API Gráfica	OpenGL, DirectX
Lenguaje de Programación	C/C++, XCode, y GCC compatible
Documentación	Sí
Sistema Operativo Soportado	Windows, OS X, Linux, and FreeBSD
Estado	Versión 7.0
Características	
Características Generales	• Diseño Orientado a Objetos
Física	• Detección de Colisiones
Iluminación	• Volumétrica
Sombra	• *Shadow Mapping, Projected Planar, Shadow Volume*
Texturizado	• Básico • Soporta imágenes JPG, PNG, BMP, PPM, PCX, TGA, DDS, e ICO
Shaders	• *Vertex*, *Pixel*, GLSL, Cg, y ASM con chequeo de errores y extensiones
Administrador de Escenas	• BSP, LOD
Animación	• Animación por *Keyframe*

Mallas	• Cargado de Mallas. Soporta 3DS, IFS, MD2, BSP, y modelos personalizables.
Efectos Especiales	• Mapeo del entorno, destellos, *Billboarding*, Sistema de Partículas, Cielo, Agua, Fuego, Explosiones, Neblina, Espejos.
Terreno	• *Rendering*.
Sistema de Red	• Cliente-Servidor, Red basada en TCP y UDP.
Inteligencia Artificial	• *Scripted*.
Rendering	• Funciones Fijas, Fuentes.
GUI	• GUI parametrizable, con Botones, Lists, Edit, Scrollbars.
Tools	• Visualizador de Modelos, GPU benchmark, Herramientas de depuración en tiempo real. • Administrador automático de memoria opcional. • Matriz n x m optimizada. • Configuración de ficheros de lectura/escritura. • Ficheros Javas y clases de red.

Tabla 2. Características de G3D.

1.4.2 Object Oriented Graphics Rendering Engine

OGRE (del inglés, *Object Oriented Graphics Rendering Engine* es un flexible motor gráfico orientado a la escena, escrito en C++, ideal e intuitivo para los desarrolladores que producen juegos y demos utilizando *hardware* gráfico. La biblioteca de clase que propone abstrae todos los detalles del uso de las API de bajo nivel como Direct3D y OpenGL, facilitando una interfaz basada en objetos del mundo virtual y otras clases muy intuitivas.

Autor	Steve Streeting
API Gráfica	OpenGL, DirectX
Lenguaje de Programación	C/C++
Documentación	Sí
Sistemas Operativos	Windows, Linux, Mac OS
Estado	Productiva/Estable
Características	
Características Generales	• Diseño Orientado a Objeto, Arquitectura de *Plugin*, Sistema de Salva/Carga. • Simple, interfaz orientada a objeto fácil de usar, diseñada para minimizar los esfuerzos necesarios para renderear escenas 3D, y para ser independiente de la implementación 3D (e.g.

	Direct3D/OpenGL). • Arquitectura flexible de plugin que permite al motor gráfico ser extendido sin realizar una recompilación. • Diseño de una documentación completa y clara de todas las clases del motor gráfico. • Soporta ZIP/PK3 para compactar.
Scripting	• Lenguaje *Scripted*.
Física	• Básica, Detección de Colisiones, Cuerpos Rígidos. • Incluye acoples para sistemas físicos como (*Open Dynamics Engine*, *Novodex* y *Tokamak*).
Iluminación	• *Per-vertex*, *Per-pixel*, *Lightmapping*. • Puede tener un número ilimitado de luces en la escena.
Sombra	• *Shadow* Mapping, *Shadow Volume*.
Texturizado	• Básico, *Multi-Texturing*, *Bumpmapping*, *Mipmapping*, *Volumetric*, *Projected*. • Soporta imágenes PNG, JPEG, TGA, BMP y DDS.
Shaders	• *Vertex*, *Pixel*, *High Level*: • Soporta *vertex* y *shaders*, ambos programas de bajo nivel escrito en ensamblador, y programas de alto nivel escrito en Cg o DirectX9 HLSL, proveyendo soporte automático para muchos parámetros constantes. • Soporta GLSL.
Administrador de Escenas	• General, BSP, *Octrees*, LOD.
Animación	• Cinemática Inversa, Animación Esquelética, Mezcla de Animación.
Mallas	• Carga de Malla, *Skinning*, *Progressive*. • *Skinning* por aceleración gráfica. • Exporta a partir de muchas herramientas de modelación que incluye Milkshape3D, 3D Studio Max, Maya, Blender y Wings3D.
Superficies & Curvas	• *Splines*.
Efectos Especiales	• *Billboarding*, Sistema de Partículas, Cielo, Agua, Neblina.
Rendering	• Funciones fijas, Fuentes, GUI. • Material LOD. • Soporte para múltiples técnicas de materiales. • Los objetos transparentes son automáticamente administrados. • Sistema de Fuentes: Fuentes *TrueType* y texturas precreadas. • Sistema GUI 2D con Botones, *Lists*, *Edit Boxes*, *Scrollbars*.

Tabla 3. Características de OGRE.

Es bueno notar que OGRE no reclama ser un *game engine* propiamente. Solo se esfuerza en hacer lo que realmente hace mejor, los gráficos. Adolece por su parte de algunas debilidades como es el caso del soporte de red tan importante hoy en día en las aspiraciones de cualquier grupo de desarrolladores en su afán de obtener juegos que cumplan con tan importante requisito. Por su parte G3D constituye una excelente biblioteca que se ajusta fácilmente a las necesidades de los programadores. Es una biblioteca de más bajo nivel que otras como la propia OGRE, así, no presenta un administrador de gráficos de la escena o de materiales, pues son cuestiones específicas de cada aplicación, pero presenta un poderoso soporte de red con el que encapsula todos los *sockets* de bajo nivel facilitando un acceso fácil a los programadores. Soporta envío de paquetes usando los dos tipos de protocolos, TCP y UDP; mecanismos de serialización que permiten la conversión de objetos a una forma que puede ser fácilmente transmitida. G3D soporta además el descubrimiento automático de servidores en la red. Lo más interesante es que de G3D puedes usar tanto como quieras, ya sea como biblioteca utilitaria o como motor gráfico. Las razones antes expuestas han hecho que muchos programadores en el mundo la prefieran, llegando a obtener productos de una alta calidad (32).

1.5 Biblioteca Física

El corazón de la simulación de un sistema articulado de cuerpos rígidos es una biblioteca física de alto rendimiento o motor físico. Los motores físicos proveen una simulación basada en la física Newtoniana para sistemas de cuerpos rígidos usando variables tales como la masa, la velocidad, la fricción y la resistencia al viento entre otras, permitiendo así simular y predecir efectos bajo diferentes condiciones, con un alto grado de aproximación a lo que ocurre en la vida real. Por lo general se clasifican en dos tipos:

1. De tiempo real.
2. De alta precisión.

Los motores físicos de alta precisión requieren más poder de procesamiento para calcular con precisión los cálculos físicos y comúnmente son utilizados por científicos en el estudio de estrategias de manipulación y control de sistemas robóticos y en películas de animados hechas en computadoras. En los videojuegos u otras formas de computación interactiva, el motor físico esta llamado a simplificar sus cálculos y a bajar su precisión de manera que pueda actuar en un tiempo razonable y apropiado durante

la ejecución del juego. Estos son los reconocidos motores físicos de tiempo real. Por tales motivos existen un sinnúmero ellos con una muy alta calidad disponibles tanto comerciales como *open source*.

1.5.1 Motores físicos de tiempo real

Open Source	***Closed Source***[5]	***Comerciales***
Open Dynamics Engine	*Newton Game Dynamics*	*Havok (propiedad de Intel)*
Bullet	*Tokamak physics engine*	*PhysX (antiguamente NovodeX incorporando Meqon)*
OPAL		*nV Physics SDK*
		Endorphin

Tabla 4. Motores físicos de tiempo real.

1.5.2 Motores físicos de alta precisión

1. Mechanica, de Parametric Technologies Corporation.
2. Working Model, de Design Simulation Technologies. (2D solamente).
3. Falling Bodies, de Animats.

Vale destacar que en febrero del 2006 se hizo el lanzamiento de la primera unidad de procesamiento físico PPU perteneciente a la Compañía Ageia, la cual con funciones como las de las tarjetas gráficas liberan al CPU del procesamiento físico. La tarjeta fue más efectiva en el aceleramiento de sistemas de partículas sin embargo la mejora del rendimiento para cuerpos rígidos fue notabilísimo también. Nvidia ha anunciado recientemente (2008) que la adquisición de Ageia es un hecho, pero si Nvidia lograra combinar esta tecnología con sus GPU podría significar un adelanto importante en el rendimiento de juegos y aplicaciones 3D. Claro que no estará solo en ese hipotético futuro mercado, pues Intel también ha adquirido Havok hace unos meses y probablemente estará planeando lo mismo. Es en este contexto donde surgen los conceptos de GPGPU (General Purpose on Graphics Processing Unit) una prometedora tecnología para motores físicos de tiempo real, que incluyen la dinámica de cuerpos rígidos y que ha llevado a asegurar a prestigiosas compañías como la ATI que su tarjeta X1900 XT debe entregar nueve veces más rendimiento que la tarjeta física Ageia PhysX. Numerosos criterios fueron evaluados en la

[5] Closed source: Distribución libre limitada.

selección del motor físico para el presente trabajo de diploma, incluyendo la existencia de una API usable, buena documentación, una activa comunidad de desarrolladores así como la disponibilidad del código fuente. *Open Dynamics Engine* (ODE) desarrollado por Russel Smith y lanzada bajo la licencia BSD de código abierto fue encontrada y resultó ser muy apropiada para estos criterios. ODE es un motor físico de completa calidad industrial que incluye numerosos tipos de articulaciones (ver figura 2), presenta una detección de colisiones integrada (aunque se le puede acoplar un sistema de colisiones propio) y una API en C/C++. En adición a numerosos juegos de computadoras comerciales, ODE ha sido utilizado en investigaciones biomecánicas y robóticas. Su ecuación para el movimiento de cuerpos rígidos es derivada del modelo de velocidad basado en los multiplicadores de Lagrange. Es rápida, robusta y estable, además el usuario tiene la completa libertad de cambiar la estructura del sistema aún cuando la simulación esté corriendo. Enfatiza velocidad y estabilidad por encima de la precisión física (33).

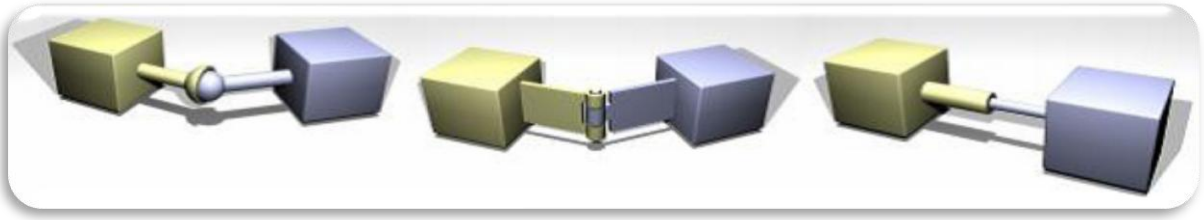


Figura 2: Tres diferentes tipos de restricciones.

1.6 Herramientas de Desarrollo

1.6.1 Microsoft Visual Studio 2005

Para el desarrollo de esta aplicación de consola Win32 escrita en C++ se seleccionó el entorno integrado de desarrollo Visual Studio 2005, dentro de este específicamente su compilador de C++ nativo. Visual Studio 2005 define un conjunto de herramientas de desarrollo profesionales para programadores individuales o para aquellos que están trabajando en pequeños equipos y que están construyendo aplicaciones. Dicha herramienta permite disfrutar de un entorno de desarrollo altamente productivo que incluye lenguajes de programación y editores de código mejorados. Dicha edición añade funcionalidad a las características básicas incluyendo herramientas para la depuración, así como un IDE completo y sin restricciones.

1.6.2 Lenguaje de Programación

Tal y como se había anunciado anteriormente se utilizará como lenguaje de programación el C++ por ser un lenguaje libre, portable y robusto soportado bajo el paradigma de la Programación Orientada a Objeto e ideal para el desarrollo de gráficos.

1.6.3 Rational Rose

Es una buena solución de modelado visual y una buena herramienta para traducir requisitos de alto nivel a una arquitectura flexible basada en componentes. Rational se encuentra a la cabeza del desarrollo del Lenguaje Unificado de Modelado (UML), que se ha convertido en la notación estandarizada empleada en Rational Rose para especificar, visualizar y construir desarrollos de *software* y sistemas. Rational Rose domina el mercado de herramientas para el análisis, diseño, modelado y construcción orientada a objetos; tiene todas las características que los desarrolladores, analistas, y arquitectos exigen: soporte UML, incomparable desarrollo basado en componentes con soporte para arquitecturas líderes en la industria y modelos de componentes, facilidad de uso e integración optimizada. Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Una de las grandes ventajas de Rose es que utiliza la notación estándar en la arquitectura de *software* (UML), la cual permite a los arquitectos de *software* y desarrolladores visualizar el sistema completo utilizando un lenguaje común. Además, los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

1.7 Actualidad Nacional

En lo que respecta a la actualidad nacional y más específicamente en nuestra Universidad podemos decir que la presente investigación constituye un tema sin precedentes aunque vale destacar la actividad del grupo de Sistemas Adaptativos del ICIMAF que guiados por el Dr. Rolando J. Biscay Lirio comienzan a interesarse en el tema. Otros profesores como el Dr. Li-Vang Lozada Chang de la facultad de Matemática Computación de la Universidad de la Habana se inician también en el área.

CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA

Introducción

A lo largo del presente capítulo será posible conocer las características del sistema. Se comenzará conociendo detalles del objeto de automatización, así como la información que se maneja, suficientes para adentrarse en la propuesta de solución, donde será posible describir primero el modelo del negocio, con sus procesos fundamentales, los actores del mismo y sus trabajadores. Además, será posible apreciar el diagrama de casos de uso del negocio, la descripción de cada uno de ellos, sus expansiones, los diagramas de actividades culminando esta primera etapa con el modelo de objetos del negocio. En un segundo momento se especificarán los requisitos de *software*, funcionales y no funcionales, se definirán los actores del sistema así como el diagrama de casos de uso del mismo. Teniendo en cuenta el desarrollo incremental se hará una propuesta de casos de uso por ciclo, incluyendo además una descripción expandida para cada uno de ellos. Posteriormente, se analizará un modelo unificado de eventos propuesto para el lenguaje C++ sobre el cual descansa la aplicación desarrollada. Como último punto se describirá el modelo de persistencia utilizado para el almacenamiento del modelo físico creado en el entorno de modelación.

CAPÍTULO 2
CARACTERÍSTICAS DEL SISTEMA

2.1 Objeto de automatización

Con la presente investigación se pretende automatizar los procesos de creación, edición y prueba de sistemas articulados de cuerpos rígidos, mediante una herramienta capaz de perfeccionar las respuestas del mismo ante diferentes eventos, pudiéndose así exportar y utilizar por terceros, minimizando de esta manera los recursos para la creación de los mismos e incrementando la productividad de los desarrolladores.

2.2 Información que se maneja

Para la confección de los sistemas articulados de cuerpos rígidos son necesarios los modelos visuales que le darán origen a los mismos. Estos serán facilitados por los diseñadores profesionales, los que posteriormente se analizarán y se descompondrán en las primitivas básicas disponibles para entonces conformar el modelo físico.

2.3 Propuesta del sistema

El *software* que se propone presenta un conjunto de utilidades que facilitan el trabajo de los desarrolladores de los modelos físicos, en un entorno amigable y con un conjunto de herramientas que cubren las funcionalidades necesarias para la construcción de animaciones, en este primer ciclo de desarrollo orientada a la modelación de autos. Dichas bondades permitirán crear cuerpos basados en las primitivas básicas disponibles, conectarlas entre ellas mediante algún tipo de enlace permitido, trasladarlos en el entorno, rotarlos, así como establecer sus correspondientes propiedades físicas tales como: masa, coeficiente de rozamiento, coeficiente de suspensión. Funcionalidades imprescindibles en cualquier editor como las de deshacer o rehacer un cambio también están previstas en este *software*. A su vez, será posible salvar el sistema articulado creado para que pueda ser reutilizado por terceros o para futuras modificaciones.

2.4 Modelo del Negocio

2.4.1 Descripción del modelo del negocio

En el campo del desarrollo de *software* resulta útil la creación de modelos que organicen y presenten los detalles importantes de problemas reales que se vinculan con el sistema informático a construir. Estos

modelos deben cumplir una serie de propiedades, entre ellas la de ser coherentes y relacionados. Uno de los modelos útiles previo al desarrollo de un *software* es el modelo del negocio. El modelado del negocio es una técnica para comprender los procesos del negocio de la organización. Los propósitos que se persiguen al realizarse el modelado del negocio son:

- Entender la estructura y la dinámica de la organización.
- Entender los problemas actuales e identificar mejoras potenciales.
- Asegurarse de que los clientes, usuarios finales y desarrolladores tienen una idea común de la organización.
- Derivar los requerimientos del sistema a partir del modelo de negocio que se obtenga.

A continuación, se describirán los principales procesos del negocio que se llevan a cabo durante la confección de los modelos físicos en el Proyecto Juegos Consola, específicamente relacionados con la modelación de autos.

1- Crear un sistema articulado

La dirección del proyecto realiza la solicitud de creación de un sistema articulado al desarrollador de los mismos, para un modelo 3D suministrado por los diseñadores. El desarrollador recibe la solicitud y el modelo 3D, descomponiendo este último en primitivas básicas. Una vez completado este paso, ubica las mismas en el espacio 3D para luego situar y fijar las uniones pertinentes que le darán consistencia al modelo, permitiendo además la inclusión de varias propiedades físicas que le darán el realismo esperado. La ubicación de las distintas primitivas constituye el paso fundamental en dicho proceso, pues garantiza en primer lugar la representación adecuada del modelo 3D a la vez que facilita la actividad de situar y fijar las uniones. Todas estas actividades antes mencionadas se realizan mediante un proceso abrumador de prueba y error ejecutado mediante instrucciones de código.

2- Editar un sistema articulado

Una vez creado el sistema articulado para un modelo 3D es posible que el último sufra cambios a menor o mayor escala. Dicha solicitud de cambio es llevada a cabo por la dirección del proyecto, la cual ordena la edición de un modelo a los diseñadores. Este modelo editado es entregado posteriormente al desarrollador del modelo físico, quien analiza la nueva composición con el objetivo de establecer los cambios, mediante el mismo proceso abrumador de situar correctamente en el espacio las primitivas finales que conformarán el modelo 3D.

2.4.2 Casos de uso del negocio

Se define como actor del negocio a cualquier individuo, grupo, entidad, organización, máquina o sistemas de información externos, con los que el negocio interactúa. Lo que se modela como actor es el rol que se juega cuando se interatúa con el negocio para beneficiarse de sus resultados (34).

Por su parte los trabajadores del negocio son una abstracción de una persona (o grupo de personas), una máquina o un sistema automatizado, que actúa en el negocio realizando una o varias actividades, interactuando con otros trabajadores del negocio y manipulando entidades del negocio. Representa un rol (34).

Teniendo en cuenta lo anterior, se definen a continuación los actores y trabajadores del negocio.

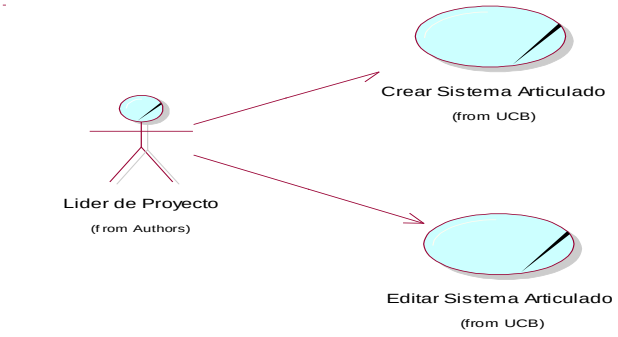
Actores del negocio	Justificación
Líder de Proyecto	Máxima autoridad del proyecto encargada de ordenar primero, la creación de un modelo 3D y luego, la construcción de su sistema articulado correspondiente. Ordena además los procesos de edición.

Trabajadores del negocio	Justificación
Desarrollador del modelo físico	Es el especialista en la construcción de sistemas articulados para la representación física de un modelo 3D suministrado.

2.4.3 Modelo de Casos de Uso del Negocio

El modelo de Casos de Uso del Negocio es un modelo que describe los procesos de un negocio (casos de uso del negocio) y su interacción con elementos externos (actores), tales como socios y clientes, es decir, describe las funciones que el negocio pretende realizar y su objetivo básico es describir cómo el negocio es utilizado por sus clientes y socios (35).

Diagrama de Casos de Uso del Negocio



CAPÍTULO 2
CARACTERÍSTICAS DEL SISTEMA

<table>
<tr><th colspan="2">Caso de Uso del Negocio: Crear Sistema Articulado</th></tr>
<tr><td colspan="2">Actores del negocio: Líder de Proyecto (Inicia)</td></tr>
<tr><td colspan="2">Propósito:
Crear un sistema articulado a partir de la descomposición en primitivas básicas de un modelo 3D suministrado.</td></tr>
<tr><td colspan="2">Caso de uso asociado:</td></tr>
<tr><td colspan="2">Resumen:
El caso de uso se inicia cuando el Líder de Proyecto le solicita al desarrollador de los modelos físicos la construcción de uno, para un modelo 3D en cuestión, previamente suministrado a él por los diseñadores profesionales. Dicho modelo es entonces descompuesto por el desarrollador del modelo físico en primitivas básicas y ubicadas estas en el espacio, de manera tal que representen correctamente el modelo 3D deseado. Una vez cumplido este paso se fijan entre las partes las correspondientes uniones y sus propiedades, quedando creado el modelo físico, basado en un sistema articulado de cuerpos rígidos. Luego se entrega el producto final. El proceso anteriormente descrito se realiza íntegramente mediante instrucciones de código.</td></tr>
<tr><td colspan="2">Curso normal de los eventos:</td></tr>
<tr><th>Acción del actor</th><th>Respuesta del negocio</th></tr>
<tr><td>1) El Líder de Proyecto hace la solicitud para la creación del modelo físico correspondiente a un modelo 3D.</td><td>2) El desarrollador del modelo físico recibe la solicitud y el modelo 3D.
3) El desarrollador del modelo físico descompone el modelo 3D en primitivas básicas mediante instrucciones de código.
4) El desarrollador del modelo físico sitúa las primitivas en las que se descompone el modelo en el espacio, mediante instrucciones de código.
5) El desarrollador del modelo físico fija las uniones correspondientes ajustando sus propiedades, mediante instrucciones de código.
6) El desarrollador del modelo físico salva para un fichero el sistema articulado creado, mediante instrucciones de código.
7) El desarrollador del modelo físico entrega el modelo físico creado basado en un sistema articulado.</td></tr>
<tr><td colspan="2">Prioridad:
Este proceso es de importancia alta.</td></tr>
<tr><td colspan="2">Mejoras:
La realización de este caso de uso a través de una herramienta de edición facilitaría los procesos de creación así como la ubicación de las primitivas en el espacio 3D.</td></tr>
<tr><td colspan="2">Cursos Alternos:</td></tr>
</table>

Caso de Uso del Negocio: Editar Sistema Articulado
Actores del negocio: Líder de Proyecto (Inicia)
Propósito: Editar un sistema articulado creado.
Caso de uso asociado:
Resumen:

El caso de uso se inicia cuando el Líder de Proyecto solicita al desarrollador de los modelos físicos la edición de un modelo 3D en cuestión, facilitando para esto la nueva versión de dicho modelo. El mismo, es entonces editado mediante la adición o sustracción de alguna primitivas básicas, ubicando las mismas en el espacio, de manera tal que representen correctamente el modelo 3D deseado. Una vez cumplido este paso se fijan entre las partes las correspondientes uniones y sus propiedades quedando editado el modelo físico basado en un sistema articulado de cuerpos rígidos. Luego se entrega el producto final. Dicho proceso se realiza íntegramente mediante instrucciones de código.

Curso normal de los eventos:

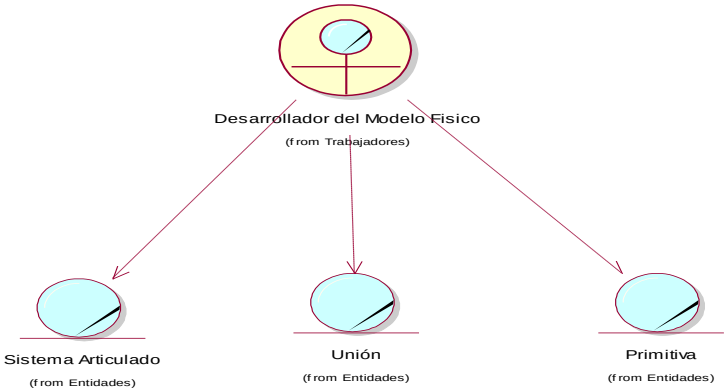
Acción del actor	Respuesta del negocio
1) El Líder de Proyecto hace la solicitud para la edición de un sistema articulado creado, a partir de una nueva versión de un modelo 3D.	2) El desarrollador del modelo físico recibe la solicitud y el nuevo modelo 3D. 3) Analiza la composición del nuevo modelo. 4) Sitúa las nuevas primitivas mediante instrucciones de código si existen. 5) Elimina las viejas primitivas mediante instrucciones de código, si existen. 6) El desarrollador del modelo físico fija las uniones correspondientes a las nuevas primitivas ajustando sus propiedades, mediante instrucciones de código. 7) Salva el Sistema Articulado, mediante instrucciones de código. 8) Entrega el modelo físico creado basado en un sistema articulado.

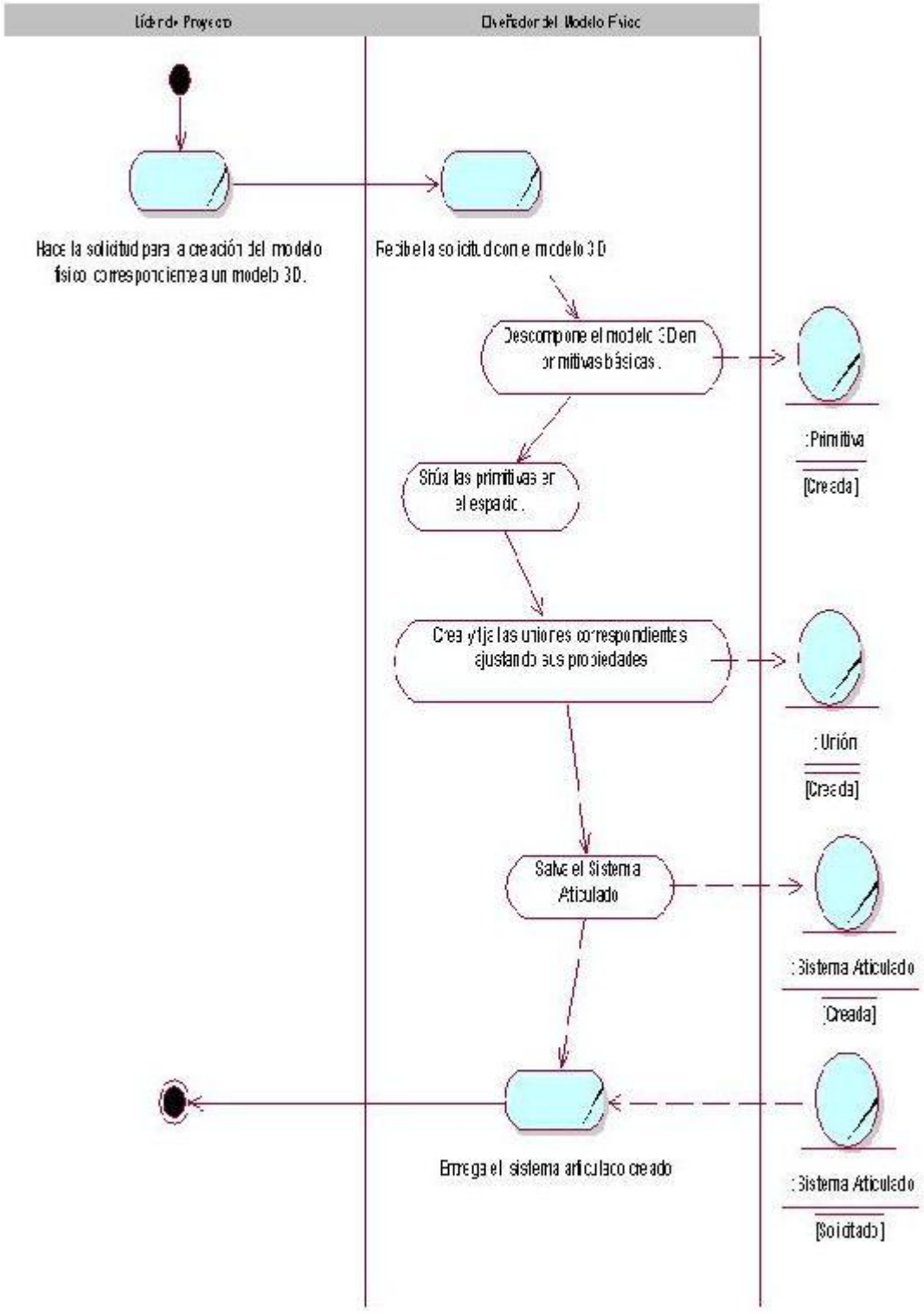
Prioridad:
Este proceso es de importancia alta.

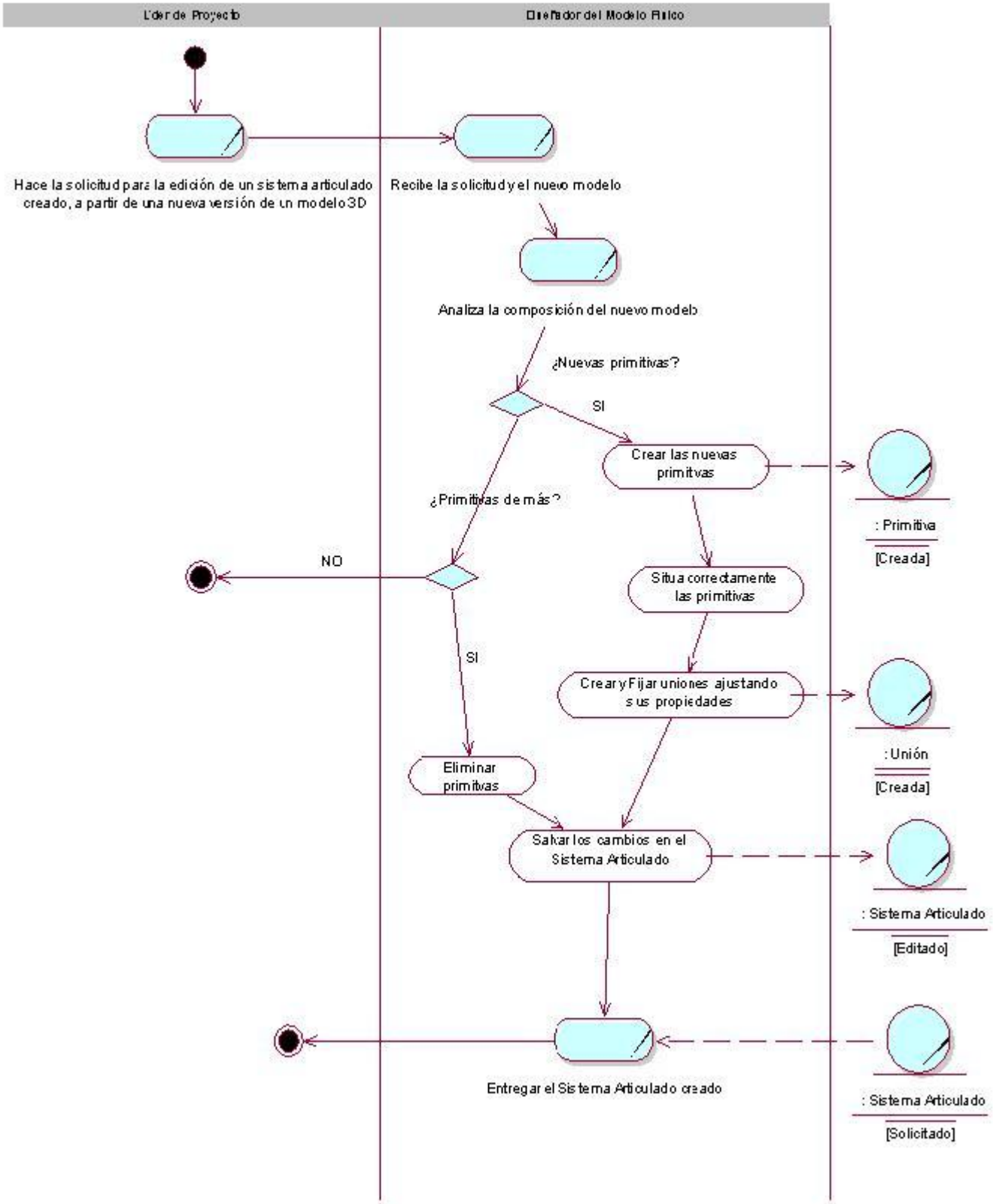
Mejoras:
La realización de este caso de uso a través de una herramienta, facilitaría los procesos de edición incrementando la productividad de los desarrolladores.

Cursos Alternos:

Modelo de Objetos del Negocio




Líder de Proyecto
Diseñador del Modelo Físico
Hace la solicitud para la creación del modelo físico correspondiente a un modelo 3D.
Recibe la solicitud con el modelo 3D
Descompone el modelo 3D en primitivas básicas.
:Primitiva
[Creada]
Sitúa las primitivas en el espacio.
Crea y fija las uniones correspondientes ajustando sus propiedades
:Unión
[Creada]
Salva el Sistema Articulado
:Sistema Articulado
[Creada]
Entrega el sistema articulado creado
:Sistema Articulado
[Solicitado]


Líder de Proyecto
Diseñador del Modelo Físico
Hace la solicitud para la edición de un sistema articulado creado, a partir de una nueva versión de un modelo 3D
Recibe la solicitud y el nuevo modelo
Analiza la composición del nuevo modelo
¿Nuevas primitivas?
SI
Crear las nuevas primitvas
: Primitiva
[Creada]
¿Primitivas de más?
NO
Situa correctamente las primitivas
SI
Crear y Fijar uniones ajustando sus propiedades
: Unión
[Creada]
Eliminar primitvas
Salvar los cambios en el Sistema Articulado
: Sistema Articulado
[Editado]
Entregar el Sistema Articulado creado
: Sistema Articulado
[Solicitado]

CAPÍTULO 2
CARACTERÍSTICAS DEL SISTEMA

2.5 Especificación de los requisitos de *software*

El análisis de requisitos permite al ingeniero de sistemas especificar la función y el rendimiento del *software*, indica la interfaz del mismo con otros elementos del sistema y establece las restricciones que debe cumplir éste. El análisis de requisitos permite al ingeniero del *software* (a menudo llamado analista en esta función) refinar la definición del *software* y construir los modelos de los dominios de datos, funcional y de comportamiento que van a ser tratados por el *software*. El análisis de requisitos proporciona modelos al diseñador del *software* que pueden traducirse en el diseño de datos, arquitectónico y de interfaz. Finalmente, la especificación de requisitos proporciona al diseñador y al cliente los medios para valorar la calidad una vez que se ha construido el *software* (35).

2.5.1 Requerimientos funcionales

Los requerimientos funcionales no son más que la determinación clara y concisa de qué debe ser capaz de hacer el sistema, éstas se corresponden con opciones que ejecutará el *software*, operaciones realizadas de forma oculta o condiciones extremas a determinar por el sistema.

1- Crear nuevo documento.
2- Crear primitivas geométricas.
    a. Crear caja.
    b. Crear cilindro.
3- Crear unión.
    a. Crear unión fija
    b. Crear unión con suspensión.
4- Seleccionar primitivas geométricas.
5- Seleccionar uniones.
6- Deseleccionar primitivas geométricas.
7- Deseleccionar uniones.
8- Eliminar primitivas geométricas.
9- Eliminar uniones.
10- Editar primitivas geométricas.
    a. Configuración de parámetros físicos y gráficos de una caja.
    b. Configuración de parámetros físicos y gráficos de un cilindro.
11- Edición de uniones
    a. Configuración de parámetros físicos y gráficos de una unión fija.
    b. Configuración de parámetros físicos y gráficos de una unión con suspensión.
12- Interfaz para configurar la posición y orientación de las geometrías y uniones.
13- Interfaz par configurar los parámetros globales del entorno físico.
14- Simular un sistema articulado.

15- Configurar eventos de teclado para que el usuario tenga control durante la simulación.
16- Selección y configuración de vista de simulación.
17- Cargar un sistema articulado.
18- Salvar un sistema articulado.
19- Deshacer un cambio.
20- Rehacer un cambio.
21- Detener la simulación.

2.5.2 Requerimientos no funcionales

Los requerimientos no funcionales responden a cualidades que el producto debe tener y a las características para que este sea atractivo, confiable, usable, seguro.

Seguridad:
El sistema debe realizar el tratamiento de errores para todas las acciones que se realizan en el sistema.

Apariencia:
El Producto debe tener una apariencia profesional. Debe ser ágil, agradable, amigable, muy legible y simple de usar, teniendo en cuenta las características de los usuarios hacia los cuales va dirigido el mismo.

Usabilidad:
El sistema será ampliamente utilizado por los diseñadores de modelos físicos. Poseerá una ayuda sensible de contexto referente de cada una de sus funcionalidades.

Rendimiento:
Para un rendimiento óptimo se crearán interfaces a pantalla completa así como varias funcionalidades para esconder las barras de edición, pudiéndose aprovechar al máximo el área de la pantalla del monitor. Además el uso de accesos directos mediante teclado permitirá habilitar de una forma rápida las funcionalidades más usadas. El sistema debe poseer un tiempo de respuesta rápido ante cualquier petición del usuario.

Soporte:
El producto debe ser fácilmente extensible para soportar nuevas funcionalidades o utilidades.

Portabilidad:

El Producto creado debe estar concebido para ser multiplataforma con ninguno o pocos cambios en su código fuente. Todas las herramientas de soporte utilizadas para la construcción del mismo deberán cumplir con dicha restricción.

Confiabilidad:

El *software* deberá poseer un número de fallos mínimos, recuperándose rápidamente de cualquier situación comprometedora. Las operaciones realizadas con éste deberán gozar de una alta precisión, garantizando la conformidad del cliente.

***Software y Hardware*:**

Vale destacar que la versión actual fue compilada con Visual Studio 2005, por lo que está actualmente disponible para Windows. No obstante, su código fuente solo tiene que ser compilado en GNU\Linux para que pueda ser utilizado en el mismo.

Esta aplicación requiere:

Microprocesador	Sistema operativo	RAM
Pentium III (mínimo).	Windows.	64MB (recomendado).

2.6 Definición de los actores del sistema

Un actor del sistema no es parte del mismo, es un rol de un usuario que puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un *software* o a una máquina que interactúa con el sistema (34). Basado en esto fue definido el actor involucrado en los casos de uso.

Actores	Justificación
Diseñador del modelo físico	Es el especialista en la construcción de sistemas articulados y el encargado de atender las solicitudes relacionadas con la construcción y edición de un modelo físico.

2.7 Casos de uso del sistema

Un caso de uso del sistema es un documento narrativo que describe la secuencia de un actor (agente externo) que utiliza un sistema para completar un proceso (34). A continuación se expondrá el listado de

casos de uso del sistema, que incluye su actor, una breve descripción de cada uno así como una referencia a su requisito funcional asociado.

CU-1	Crear Nuevo Documento
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico crea un documento en blanco donde se construirá el nuevo sistema articulado.
Referencia	RF: 1

CU-2	Crear Primitiva Geométrica
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico crea una primitiva geométrica partiendo de la configuración de sus propiedades tanto físicas como gráficas.
Referencia	RF: 2

CU-3	Crear Unión
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico crea una unión entre dos geometrías (previamente seleccionadas) partiendo de la configuración de sus propiedades tanto físicas como gráficas.
Referencia	RF: 3

CU-4	Seleccionar Objetos de la Escena
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico selecciona las primitivas geométricas o las uniones deseadas para realizar posteriormente alguna acción sobre estas.
Referencia	RF: 4, 5

CU-5	Deseleccionar Objetos de la Escena
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico deselecciona las primitivas geométricas o las uniones deseadas para descartar alguna acción sobre estas.
Referencia	RF: 6, 7

CU-6	Eliminar Objetos de la Escena
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico elimina las primitivas geométricas o las uniones que hayan sido seleccionadas.
Referencia	RF: 8, 9

CU-7	Editar Primitiva Geométrica
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico edita las propiedades tanto físicas como gráficas de una geometría seleccionada.
Referencia	RF: 10, 10.1, 10.2

CU-8	Editar Unión
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico edita las propiedades tanto físicas como gráficas de una unión previamente seleccionada.
Referencia	RF: 11, 11.1, 11.2

CU-9	Configurar Orientación y Posición
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico a través de un manipulador amigable configura en tiempo real tanto la posición como la orientación de las primitivas en la escena.
Referencia	RF: 12

CU-10	Configurar Propiedades del Entorno
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico a través de un formulario establece los parámetros físicos globales que regirán en el mundo virtual donde más tarde se simulará el sistema articulado.
Referencia	RF: 13

CU-11	Simular Sistema Articulado
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico introduce el sistema articulado en un entorno cargado de disímiles eventos que permitirán hacerle pruebas al mismo. Durante esta operación será posible manipular el sistema a través de eventos predeterminados de teclado a la vez que será posible cambiar la vista de simulación.
Referencia	RF: 14, 15, 16

CU-12	Cargar Sistema Articulado
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico carga desde un fichero un sistema articulado, introduciendo el mismo en el entorno de modelación 3D.
Referencia	RF: 17

CU-13	Salvar Sistema Articulado
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico salva para un fichero un sistema articulado. Dicha operación incluye cada una de sus propiedades físicas y gráficas.
Referencia	RF: 18

CU-14	Deshacer un cambio
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico deshace un cambio realizado, seleccionando la opción de deshacer o a través de la clásica combinación *control + z*.
Referencia	RF: 19

CU-15	Rehacer un cambio
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico rehace un cambio deshecho, seleccionando la opción de rehacer o a través de la clásica combinación *control + y*.
Referencia	RF: 20

CU-16	Detener la simulación
Actores	Diseñador del modelo físico.
Descripción	El diseñador del modelo físico detiene la simulación en curso, seleccionando para ello la opción de *detener simulación,* disponible en la barra de herramientas de la aplicación.
Referencia	RF: 21

2.7.1 Diagramas de casos de uso

Basándose en que un diagrama de casos de uso representa un conjunto de casos de uso para un sistema, los actores y la relación entre casos de uso y actores, se realizó entonces el diagrama de casos de uso que se muestra a continuación.

Diagrama de casos de usos del sistema Anexo 1.

2.7.2 Casos de usos por ciclos

Teniendo en cuenta el desarrollo incremental e iterativo que propone la metodología RUP, o sea, que este se realice por ciclos, en cada uno de los cuales se agregue una funcionalidad adicional, se definieron 3 ciclos de desarrollo.

En el Anexo 2 se encuentran los casos de uso por ciclos.

2.7.3 Casos de usos expandidos

A través de la expansión de los casos de uso se describe paso a paso la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema.

En el Anexo 3 se puede apreciar la expansión de los casos de uso del sistema.

2.8 Modelo unificado de eventos para el lenguaje C++ nativo, independiente del entorno de desarrollo

En el pasado, distintos entornos de programación proporcionaban medios independientes para que los componentes devolvieran a los clientes información acerca de eventos asincrónicos. El lenguaje C incluye devoluciones de llamada a funciones que realizan esto, pero a C++ siempre le ha faltado este tipo de devolución de llamada para los métodos de los objetos. Así, es posible perder mucho tiempo tratando de suplir esta falta en C++ (mediante mensajes de ventana, interfaces de eventos, nombres de método de devolución de llamada especificados en el código).

La finalidad del *Modelo unificado de eventos para el lenguaje C++ nativo, independiente del entorno de desarrollo* es permitir a las aplicaciones utilizar eventos minimizando el conocimiento y las dependencias que tiene un componente en sus clientes y maximizando la capacidad de reutilización del mismo, de forma coherente, con clases de C++ nativo e independiente del entorno de desarrollo. El modelo permite derivar subclases a partir de clases de origen de eventos o de receptores de eventos y ofrece una mayor compatibilidad con la emisión y la recepción de eventos en la clase derivada. Dicho modelo descansa sobre una arquitectura fácilmente extensible.

2.8.1 Elementos del control de eventos. Conceptos fundamentales

- Un *origen de eventos* es un objeto que define y contiene eventos.
- Un evento es un método contenido en un origen de eventos que, al ser llamado, genera eventos.
- Un *delegado* es una clase que puede contener una referencia al método. Una clase delegada difiere de las demás clases en que tiene una firma y solo puede contener referencias a métodos que coincidan con su firma.
- Un *receptor de eventos* es un objeto que recibe eventos.

- Un *controlador de eventos* es un método de un receptor de eventos que recibe eventos.

Vale destacar que con el entorno integrado de desarrollo Visual Studio (VS) es posible implementar buenos mecanismos de manipulación de eventos, puesto que provee un conjunto de atributos, palabras reservadas y funciones intrínsecas, capaces de abstraer a los desarrolladores, de las rutinas relacionadas con los punteros a funciones los cuales constituyen el fundamento de todo este mecanismo (36). Pero todas estas bondades se encuentran atadas al entorno de desarrollo VS, cuestión que las descarta si analizamos el hecho de que se persigue realizar una implementación portable e independiente del entorno de implementación.

Por su parte Borland C++ Builder constituye otro poderoso entorno de desarrollo que ha contribuido al lenguaje C++ con extensiones factibles para la manipulación de eventos; representa hoy un entorno de implementación notable al hablar del mecanismo de manipulación de eventos de sus componentes. Pero si un desarrollador desea crear sus propias rutinas de manipulación es preciso que en C++ Builder se acuda al mecanismo de los punteros a funciones a pesar de que da soporte con algunas palabras reservadas como __*closure*, con la que es posible asignar un manipulador de evento al Inspector de Objetos del entorno o asignar una función miembro de otra clase a una propiedad de un componente (37). Aún cuando también el entorno Borland C++ Builder contribuye con sus propias extensiones al desarrollo de mecanismos de manipulación de eventos, las mismas adolecen del hecho de que también se encuentran atadas el entorno de implementación, imposibilitando la reutilización del código en otros entornos o plataformas.

Las circunstancias antes enunciadas, son la causa de que el presente trabajo de diploma se trace como meta complementaria la implementación de un mecanismo unificado de manipulación de eventos para el lenguaje C++ nativo, capaz de abstraer el trabajo de los desarrolladores, de rutinas engorrosas relacionadas con punteros a funciones, a la vez que cumple con los requisitos de reutilización antes expuestos. La facilidad de uso al estilo de potentes lenguajes de alto nivel como C# es también una característica del modelo unificado propuesto, que sin dudas contribuye positivamente sobre cualquier grupo de desarrollo al hablar del incremento de la productividad que se logra con éste.

2.8.2 Especificación del modelo unificado

Las GUI actuales brindan la posibilidad de poder responder con una acción ante determinado evento que ocurra. Como se dijo anteriormente, existen muchos mecanismos para poder especificar la tarea a

realizar en caso de que ocurra determinado evento, y en cada plataforma existe una manera de lograrlo programáticamente o por ayuda del entorno de programación utilizado. Para el desarrollo de la aplicación se utilizará el sistema GUI de G3D 7.0 que presenta un conjunto de componentes gráficos aún en desarrollo. Los componentes visuales tienen a lo sumo punteros a funciones para manipular los eventos, por lo que habría que manipular cada componente de una manera muy particular, resultando ésta en una forma poco intuitiva.

Es por esto que se ha decido implementar un módulo que soporte manipular, tanto los eventos generados por G3D 7.0, como los nuevos eventos que tengan que ser definidos por la creación de nuevos componentes, o incluso llevar a cabo acciones lógicas en caso de colisiones físicas. En este módulo, es muy sencillo crear o invocar delegados y además es muy fácil crear nuevos tipos de delegados. Los delegados cuentan con un recolector de basura el cual libera de responsabilidad al programador de tener que destruir los objetos creados, todo esto de una manera eficiente.

El modelo de delegados es en fin un módulo para poder abstraer la complejidad de los eventos y las distintas acciones que se necesiten tratar a lo largo del desarrollo del Producto.

2.8.3 Filosofía de uso

Existen muchos tipos de eventos tales como: eventos de teclado, eventos de mouse, eventos de botones, de selección. Antes de hacer un delegado es necesario pensar qué información va a recibir y hacer un tipo de datos que contenga los argumentos del evento. Por ejemplo, para manipular eventos de G3D el tipo de dato utilizado es **G3DEvent**. En esta estructura se pondrán todos los datos necesarios para manipular un evento de G3D.

Para construir un delegado es necesario asignarle un método *(un controlador).*

Como método de clase es necesario especificar los tipos de datos que va a recibir *(firma)*. En la aplicación todos los eventos van a ser tratados con un único tipo de controlador:

```
void método(void* sender, const <Arg>& e)
```

Note que *Arg* esta subrayado entre paréntesis angulares pues aquí se espera un tipo de dato cualquiera, que sería el argumento del evento.

Hacer un *controlador* es simplemente declarar e implementar un método en una clase que constituye el evento.

```
class MyClass
   {
      ….
      void handle1(void* sender, const G3DEvent& e);
      ….
   };
```

Este controlador se implementaría como cualquier método de dicha clase. Después que se tiene el controlador todo esta preparado para crear el delegado, de la siguiente manera.

```
MyClass myClass;
DelegateRef<G3DEvent>  d = DELEGATE(&myClass,&MyClass::handle1);
```

Es decir, se crea la instancia del objeto y se crea el delegado pasando por parámetro el objeto receptor y el controlador. Es necesario que los argumentos del delegado (en este caso G3DEvent) sea igual a la firma del controlador. Note que un delegado no depende del tipo de clase donde este contenido el controlador aunque si hay que especificarle el objeto receptor al cual pertenece. Esta característica es la principal ventaja y será explicada en la sección *MulticastDelegate***.**

Después de este paso se puede invocar el delegado muy fácilmente, tal y como se describe a continuación:

```
…. //Establecer el objeto que inicia el evento

void* sender = obj;

G3DEvent g3dEvent;

…//Establecer los parámetros del evento y llenarlo

d->invoke(sender,arg).   //Ejecutar el evento.
```

No es necesario pensar en la destrucción de los eventos pues se destruyen una vez que se detecta que nadie los está referenciando.

2.8.4 MulticastDelegate

En muchísimas ocasiones es necesario ejecutar varias acciones dado un mismo evento. Esto no es posible con un solo delegado; para ello sería necesaria una lista de delegados pertenecientes a un mismo evento. Es por esto que *MulticastDelegate* existe. Un *MulticastDelegate* es básicamente una lista de delegados. La única condición que deben cumplir los delegados asociados a un *MulticastDelegate* es que tengan la misma firma; condición que es muy lógica ya que está asociado a un evento en particular. Por tanto, en un *MulticastDelegate* puede haber delegados con controladores pertenecientes a distintas clases. Básicamente un *MulticastDelegate* es un contenedor donde se suscriben objetos receptores y donde todos ponen la acción a tomar en caso de que ocurra el evento. Más tarde cuando el evento es detectado y ejecutado el *MulticastDelegate* se encarga de llamar a todos los que se han suscrito para invocar sus acciones.

Su creación es muy simple:

```
MulticastDelegate<G3DEvent> multicast;
multicast.add(d);
multicast.remove(d);
multicast.invoke(sender,arg);
```

Vale destacar que no tiene recolector de basura. Se le pueden agregar y quitar eventos, además de poderse tratar como delegados al ser invocados. Al ser invocado un *MulticastDelegate* este bloquea la lista de delegados inscritos, de modo que las modificaciones a esta lista no tienen efecto hasta que se termina de ejecutar completamente. Después de esto se aplican las modificaciones. También tiene sobrecarga de operadores += y -= para comodidad.

Es necesario agregar, que el simple hecho de declarar un delegado o un *MulticastDelegate* no significa que se va a ejecutar cada vez que ocurra el evento; pero sí es una manera de dejar plasmado lo que se quiere hacer cuando suceda el mismo.

2.9 Modelo de persistencia

Una vez conformado el sistema articulado de cuerpos rígidos en el editor, se hace necesario almacenar el mismo utilizando un mecanismo de persistencia, específicamente un fichero, con el objetivo de que pueda ser reutilizado por terceras personas y de una manera eficiente. El mecanismo de persistencia utilizado descansa sobre las API de alto nivel para tratamiento de ficheros que provee G3D como biblioteca

utilitaria, y en las que resulta fácil imitar estrategias de serialización y deserialización (38). Con el mecanismo propuesto por estas estructuras se humaniza notablemente el trabajo con los ficheros en C++, las mismas brindan un conjunto eficiente de funcionalidades que permiten hacer las más diversas operaciones con mucho menos esfuerzo, al decir de su poder de encapsulamiento.

El fichero a utilizar será binario pues teniendo en cuenta los niveles de complejidad que pueden llegar a tener los modelos creados con la aplicación, su utilización implicaría mayor velocidad de cómputo, pues los datos se almacenan de la misma forma en que son representados en memoria. Contrario a esto los ficheros de texto almacenan los datos en forma de caracteres, lo cual implica que deban ser convertidos a su forma binaria, descartándolos para este propósito. Un factor importante resulta además el hecho de que con los ficheros binarios se aumentarían los niveles de confidencialidad de la información del archivo, dificultándose algún tipo de modificación efectiva fuera del entorno de modelación. Vale destacar que el formato del fichero presenta una estructura fácilmente extensible basada en regiones tales como cabecera, índice y cuerpo que permiten de manera eficiente la reutilización de ficheros de versiones anteriores en entornos con versiones superiores a éste, y viceversa. La extensión del mismo es **.ras**, del Inglés *Rigid Articulated System*, por ser el acrónimo relativo a su contenido.

2.9.1 Estructura Interna

Tipos de datos del fichero

Los tipos de datos encontrados en el fichero pueden ser primitivos (propios del lenguaje), o compuestos (propios de la biblioteca G3D). Para la eficiente creación del fichero se garantizaron rutinas de serialización y deserialización para cada uno de ellos. A continuación se relacionan los mismos en la siguiente tabla.

Tipo de Dato	Descripción	Representación en el fichero
string	Cadena de caracteres: primitivo	nombre, versión, comentario, nombGeom1, nombGeom2, nombAxis, nombSteeringAxis
double	Valor de punto flotante de doble precisión: primitivo.	longitud, radio, cfm, erp, anguloMin, anguloMax
Vector3	Vector de 3 dimensiones (x, y, z): compuesto.	dimensiones
CoordinateFrame	Matriz de 4x4 utilizada para las transformaciones de rotación y traslación en los cuerpos rígidos:	cf

	compuesto.	
Color3	Color de 3 componentes (R, G, B): compuesto.	color

Tabla 5. Tipos de datos del fichero.

Especificación de un Bloque de Información

La información relativa a un objeto creado en el editor constituye un bloque de información y queda definido de la siguiente manera:

1. Un código identificador para el objeto
2. Longitud en byte del objeto.
3. Atributos del objeto.

Apariencia de un bloque de información.

```
Tipo_de_objeto  longitud  atributo1, atributo2,....
```

Los códigos para los tipos de objeto se relacionan en la siguiente tabla

TIPO	CÓDIGO
BOX	0
CYLINDER	1
FIXEDJOINT	3
STEERINGWHEELJOINT	4
AXIS	5
STEERINGAXIS	6

Tabla 6. Especificación de tipos con sus códigos.

Vale destacar que las cantidades de atributos pueden variar según el tipo de objeto, tal y como se muestra en la siguiente especificación.

Tipos de bloques según el objeto.

```
BOX nombre color cf dimensiones
CYLINDER nombre  color  cf  longitud  radio
FIXEDJOINT nombre color  nombGeom1  nombGeom2
STEERINGWHEELJOINT  nombre  color  nombGeom1  nombGeom2  nombsteeringAxis
nombAxis cfm erp
AXIS nombre color cf
STEERINGAXIS nombre color cf anguloMin anguloMax
```

Cabecera

El fichero cuenta con una cabecera, la cual constituye un bloque de metadatos útiles para interpretar correctamente la información contenida. Dicha cabecera contiene:

1. Un código identificador de cabecera
2. La longitud en bytes del bloque cabecera
3. El nombre con el que se salvo el fichero.
4. La versión del fichero.
5. La cantidad de bloques de información.
6. Una región de comentario para incluir alguna información de interés.

Apariencia de la cabecera del fichero:

```
Cabecera longitud
      nombre
      versión
      cantidad_de_regiones
      comentario
```

Índice

Además de la cabecera, el fichero cuenta a continuación de ésta con un índice, en el cual quedan perfectamente definidas las localizaciones de cada una de las regiones del cuerpo del fichero. El propósito del mismo es que quien vaya a utilizar el fichero pueda leer exactamente lo que desea sin tener que revisar información irrelevante para él, además de poder aprovechar al máximo las potencialidades de los ficheros binarios. El formato del índice se define como sigue:

1. Un código identificador para el índice.
2. La longitud en bytes del bloque índice.
3. Cada una de las regiones del cuerpo del fichero con sus respectivas posiciones en bytes *relativas* a este.

Apariencia del índice del fichero.

```
Índice longitud
   Región_de_objetos posición
      .
      .
      .
```

Cuerpo

La definición del cuerpo le sigue a la especificación del índice y su objetivo es almacenar todas las diferentes regiones que se definan con sus respectivos objetos dentro. El cuerpo representa el corazón de la información almacenada en el fichero y se define siguiendo las siguientes reglas:

1. Un código identificador para el cuerpo.
2. El código de la región en turno y su longitud.
3. Los bloques de información relativas a la región.

Apariencia del cuerpo del fichero.

```
Cuerpo
 Región_de_Objetos  longitud
      Tipo_de_objeto  longitud  atributo1, atributo2,....
      Tipo_de_objeto  longitud  atributo1, atributo2,....
      .
      .
  .
  .
```

Vista general del fichero

```
Cabecera longitud
   nombre
   versión
   cantidad_de_regiones
   comentario
Índice longitud
   Región_de_objetos posición
   .
   .
Cuerpo
   Región_de_Objetos  longitud
      Tipo_de_objeto  longitud  atributo1, atributo2,....
      Tipo_de_objeto  longitud  atributo1, atributo2,....
      .
      .
   .
   .
```

CAPÍTULO 3. ANÁLISIS Y DISEÑO DEL SISTEMA

Introducción

El análisis y el diseño constituyen partes fundamentales dentro del proceso de desarrollo de *software*. El primero tiene como propósito primario formular el modelo de dominio del problema permitiendo responder la pregunta de qué hacer. Por su parte el diseño tiene como objetivo decidir cómo el sistema se llevará a cabo. Durante el mismo se toman decisiones estratégicas y tácticas para cumplir los requerimientos funcionales y de calidad de un sistema. Con él se le da respuesta a la pregunta de cómo hacer (34). Normalmente, las clases del análisis evolucionarán directamente a los elementos del modelo de diseño; algunos convertidos en clases del diseño, otros convertidos en subsistemas de diseño. El objetivo del análisis es identificar un esbozo preliminar del comportamiento requerido a partir de los elementos de modelación del sistema. El objetivo del diseño es transformar este esbozo preliminar y algunas veces idealizado en un conjunto de elementos del modelo que serán posteriormente implementados (35). Como resultado hay un refinamiento en detalle y precisión a la vez que ocurre un movimiento desde el análisis al diseño. Además, las clases del análisis son con frecuencia lo suficientemente fluidas, cambiables y evolucionan satisfactoriamente antes de fraguarse en actividades del diseño.

En el presente capítulo se plantea el análisis y el diseño del sistema, utilizando para su modelación el UML y quedando plasmados los diagramas de clases del análisis, de clases del diseño y de interacción como artefactos fundamentales en esta fase. Los mismos han sido separados por casos de uso para facilitar su comprensión.

CAPÍTULO 3
ANÁLISIS Y DISEÑO DEL SISTEMA

3.1 Modelo de Análisis

El modelo de análisis contiene las clases del análisis y todos aquellos artefactos asociados. El modelo de análisis puede ser un artefacto temporal, tal es el caso donde el mismo evoluciona hacia el modelo de diseño o cuando continúa viviendo a través del proyecto, o quizás más allá, sirviendo como un modelo conceptual general del sistema (35).

Encontrar un conjunto candidato de clases del análisis es el primer paso en la transformación del sistema desde el mero estado de comportamiento requerido a una descripción de cómo el sistema trabajará. En este esfuerzo, las clases del análisis son usadas para representar los roles de los elementos del modelo, los cuales proveerán el comportamiento necesario para satisfacer los requerimientos funcionales especificados por los casos de uso y los no funcionales especificados por los requerimientos adicionales (35). Las clases *interfaz* y de *control* típicamente evolucionan a elementos de diseño de la capa de aplicación, mientras que las clases *entidad* evolucionan a elementos de diseño específicos del dominio (35).

Más allá de brindarnos una guía específica a la hora de encontrar clases, estos estereotipos (*interfaz, control y entidad*) resultan en un modelo de objeto robusto, pues los cambios en el modelo tienden a afectar solamente un área específica. Los cambios en la interfaz de usuario, por ejemplo, afectarán solamente las clases interfaz. Los cambios en el flujo de control afectarán solamente las clases de control, mientras que los cambios en la información a largo plazo solamente afectarán las clases entidades (35).
Las clases del análisis representan un modelo conceptual anticipado para las “cosas” del sistema que tienen responsabilidades y comportamientos (35).

El modelo de clases del análisis se muestra en el Anexo 5. El mismo se presenta por casos de uso para facilitar su comprensión. Con el objetivo de abreviar se han presentado solamente aquellos diagramas de clases con estructuras diferentes, especificándose en todos los casos aquellos que están relacionados. Para el análisis, vale destacar que los diagramas de clases pertenecientes a los casos de uso *Crear Primitiva* y *Crear Unión, Editar Primitiva y Editar Unión* son semejantes respectivamente, sustituyendo en el nombre de las clases la palabra “Caja” por “Unión Fija” y la palabra “Cilindro” por “Unión Suspensión”.

Los diagramas de clases para los casos de uso Deshacer Cambio y Rehacer Cambio son idénticos. Los diagramas de clases para los casos de uso *Seleccionar Objeto* y *Deseleccionar Objeto* son semejantes sustituyendo la palabra “Seleccionar” por “Deseleciconar”.

3.2 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización de los casos de uso y sirve como una abstracción del modelo de implementación y de código fuente, pues es utilizado como una entrada esencial en las actividades de implementación y prueba (35). Es usado además para concebir en adición a la documentación, el diseño del sistema *software*. De manera general es un abarcador y compuesto artefacto que encapsula todas las clases de diseño, subsistemas, paquetes, colaboraciones, y las relaciones entre ellos.

El modelo de diseño propuesto satisface las siguientes características.

1- Satisface los requerimientos del sistema.
2- Es resistente a cambios en el entorno de implementación.
3- Es fácil de mantener en relación a otros posibles modelos de objetos y a sistemas de implementación.
4- Queda claro como implementar.
5- Se adapta fácilmente a cambios en los requerimientos.

3.2.1 Diagrama de clases

Una clase es una descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y significado semántico (35). Una clase de diseño representa una abstracción de una o más clases en la implementación de un sistema. Exactamente la correspondencia depende del lenguaje de implementación, al igual que su tamaño y sus objetos (35). Las clases definen objetos, los cuales a su vez implementan los casos de uso; si las mismas son buenas o no, dependen profundamente del entorno de implementación. De manera general las clases deben mapearse en un

fenómeno en particular en el lenguaje de implementación y deben estar estructuradas de manera que representen los resultados en un buen código.

Aún cuando las peculiaridades del lenguaje de implementación influyen en el modelo de diseño, en el presente trabajo de diploma se ha tratado de mantener una estructura de clases fácil de modificar y entender. Las mismas son mostradas en el Anexo 6, separadas por casos de uso. Con el objetivo de brindar una mejor claridad en los diagramas de clases, se han omitido las operaciones no fundamentales. Las mismas pueden ser apreciadas en los diagramas de secuencia. Con el objetivo de abreviar se han presentado solamente aquellos diagramas de clases con estructuras diferentes, especificándose en todos los casos, aquellos que están relacionados. Para el diseño, vale destacar que los diagramas de clases pertenecientes a los casos de uso *Crear Primitiva* y *Crear Unión, Editar Primitiva y Editar Unión* son semejantes respectivamente, sustituyendo en el nombre de las clases la palabra “Box” por “FixedJoint” y la palabra “Cylinder” por “SWJoint”. Los diagramas de clases para los casos de uso Deshacer Cambio y Rehacer Cambio son idénticos. Los diagramas de clases para los casos de uso *Seleccionar Objeto* y *Deseleccionar Objeto* son semejantes sustituyendo la palabra “Select” por “Deselect”.

3.2.2 Diagramas de Interacción

Los diagramas de interacción describen el modo en el que cada operación detectada en los diagramas de secuencia lleva a cabo sus responsabilidades y modifica el estado del sistema. En UML los diagramas de interacción pueden representarse a través de los Diagramas de Colaboración y/o de los Diagramas de Secuencia, ambos son representaciones alternas de interacciones. Los Diagramas de Secuencia muestran interacciones entre objetos basadas en el tiempo y los Diagramas de Colaboración muestran cómo los objetos se asocian unos con otros (35).

El tipo de diagrama seleccionado para construir los diagramas de interacción fue el de secuencia, debido a que ellos muestran cómo los objetos se comunican unos con otros en una secuencia de tiempo, así como lo que sucede en cada momento, conteniendo para ello, los objetos con sus ciclos de vida y los mensajes que se envían entre ellos ordenados secuencialmente.

Los diagramas de secuencia son particularmente importantes para los diseñadores pues clarifican los roles de los objetos en un flujo, facilitando así una entrada básica para determinar las responsabilidades de una clases y las interfaces.

A diferencia de un diagrama de colaboración, un diagrama de secuencia, incluye secuencias cronológicas, pero no incluye relaciones entre objetos. A pesar de que expresan información similar pero mostrada de diferentes maneras, los diagramas de secuencia muestran una secuencia explícita de mensajes y son mejores cuando es importante visualizar el tiempo en que son ordenados los mismos, tal y como es el caso del presente trabajo de diploma.

Los diagramas de secuencia se muestran en el Anexo 7. Con el objetivo de abreviar se han presentado solamente aquellos diagramas de secuencia con estructuras diferentes, especificándose en todos los casos aquellos que están relacionados. Vale destacar que los diagramas de secuencia de los casos de uso *Crear Primitiva* y *Crear Unión, Editar Primitiva y Editar Unión* son semejantes respectivamente, sustituyendo en el nombre de las clases la palabra “Box” por “FixedJoint” y la palabra “Cylinder” por “SWJoint”. Los diagramas de secuencia para los casos de uso Deshacer Cambio y Rehacer Cambio son semejantes cambiando los mensajes “undo” por “redo” y “unExecute” por “execute”. Los diagramas de secuencia para los casos de uso *Seleccionar Objeto* y *Deseleccionar Objeto* son semejantes sustituyendo en el nombre de las clases la palabra “Select” por “Deselect” y los mensajes “addSelection(list)” por remove(list), select() por unSelect().

3.3 Tratamiento de errores

Para el tratamiento de errores en el presente sistema, se parte de la idea de que una aplicación bien diseñada debe disminuir la posibilidad de cometer errores. En el sistema, el tratamiento de errores está enfocado principalmente a errores producto de la interacción del usuario con el sistema, que son aquellos en los que se puede incurrir a falta de conocimiento o experiencia en la explotación del mismo. Se trata en todo momento de minimizar la posibilidad de ocurrencia de errores de este tipo, aprovechando las posibilidades de la interfaz gráfica, es decir, se evita que el usuario juegue un papel más activo en la

captación de información, para lo cual se le dará la opción de elegir o seleccionar la información que se conoce, lo cual facilitará la entrada de datos y la rapidez de la misma.

Evidentemente los errores ocurrirán incluso con los usuarios de más habilidad y experiencia. En el caso de los datos que sean adicionados por un usuario del sistema, se hace una validación de estos mediante funciones que garantizan que sean válidos; en caso que ocurra un error, se le presenta una caja de diálogo con el mensaje donde se describe el mismo. Al obtener la confirmación de lectura del mensaje de error por parte de usuario, la caja del diálogo desaparece y continúa la ejecución de la aplicación.

3.4 Concepción general de la ayuda del sistema

Es necesario para los usuarios, contar con una ayuda que les permita consultar las funcionalidades de cada una de las utilidades de la herramienta. Por ello, se brinda una ayuda sensible de contexto para cada una de estas. El usuario deberá posicionar el *mouse* o ratón sobre aquella opción que desee conocer y el sistema brindará una breve descripción de su uso.

3.5 Patrones de diseño

Fachada: Provee una interfaz unificada para manipular el comportamiento de un subsistema. Se hace con el objetivo de facilitar el uso de dicho subsistema. Para la GUI de G3D 7.0 se tuvo que construir una fachada, debido a que la misma constituye una versión aún en desarrollo y su utilización resulta poco intuitiva. Utilizando este patrón se logró definir una interfaz de alto nivel que hizo más fácil el trabajo con la GUI de G3D.

Builder: Separa la construcción de un objeto complejo de su representación, de tal manera que el mismo proceso puede crear diferentes representaciones. En el caso de la aplicación, era necesario crear las interfaces de cada primitiva con sus respectivos atributos. Para esto se notaba la creación de atributos semejantes en cada primitiva, de modo que era necesario reutilizar estos atributos comunes. Entonces cada “PaneSet” es un *builder* que se encarga de generar la lista de interfaces para cada atributo que debe ser mostrado.

Command: Encapsula una petición como un objeto, permitiendo parametrizar las solicitudes de los clientes (colas, registro de solicitudes) y soportando además la cancelación de operaciones. En el caso de la aplicación, era necesario emitir una petición a un objeto sin que este conociera la operación solicitada o el receptor de la petición. Los botones y los menús de las barras de herramientas llevan a cabo solicitudes en respuesta a las entradas del usuario, pero estas no pueden implementar explícitamente las solicitudes en un botón o menú, porque solo las aplicaciones que usen la barra de herramientas deberían conocer que hacer para cada objeto. Además, el diseñador de la misma no tiene forma de saber el receptor o la operación que se llevará a cabo en cada caso. Con la utilización de este patrón se potencia la reutilización.

Memento: Sin violar el encapsulamiento, captura y externaliza el estado interno de un objeto de manera que dicho objeto puede ser restaurado a este estado posteriormente. En el caso de la aplicación, era necesario almacenar el estado interno de un objeto, para permitir al usuario retractarse de algún cambio o recuperarse de los errores. Pero los objetos normalmente encapsulan todo su estado, haciéndolo inaccesible e imposibles de externalizar. Exponer este estado violaría el encapsulamiento, lo cual compromete la fiabilidad y la extensibilidad de la aplicación. Con este patrón se resolvió el problema; un *memento* es un objeto que almacena una foto del estado interno de otro objeto, su creador. El mecanismo de cancelación solicitará un memento desde su creador cuando éste necesite controlar el estado del mismo. Así, este creador inicializa el memento con la información que caracteriza su estado actual. Solo el creador puede almacenar y recuperar información desde el memento, el memento es opaco para los demás objetos.

Singleton: En el caso de la aplicación era necesario asegurar que una clase tuviera una única instancia (e.g. estado de edición, entorno) y proveer un punto global para acceder a esta. ¿Cómo asegurar que esa clase tuviera una única instancia y que al mismo tiempo fuera fácilmente accesible? Una variable global hace un objeto accesible, pero no limita la instanciación de múltiples objetos de un tipo. Una mejor solución fue hacer que la clase misma fuera responsable de garantizar su única instancia, interceptando solicitudes para crear nuevos objetos y facilitando además una vía para acceder a la instancia. Este es el patrón Singleton.

CAPÍTULO 4. IMPLEMENTACIÓN

Introducción

El flujo de trabajo de *Implementación* se hace con el objetivo de definir la organización del código teniendo en cuenta los subsistemas de implementación organizadas por capas, la implementación de los elementos de diseño en términos de ficheros fuentes, binarios, ejecutables y para poder integrar los diferentes componentes de desarrolladores o equipos y generar un ejecutable entregable o producto final. Al finalizar, en la implementación deben quedar plasmados todos los requisitos recogidos en la fase de *Requerimientos*. Vale destacar que este flujo de trabajo está fuertemente regido por el flujo de *Análisis y Diseño.*

En este capítulo se expondrá el artefacto Modelo de Implementación, que incluye la especificación de cada uno de sus subsistemas, destacándose al final las reglas de codificación que se siguieron durante la realización de dicho flujo de trabajo.

CAPÍTULO 4
IMPLEMENTACIÓN

4.1 Modelo de Implementación

El objetivo del modelo de implementación es identificar las partes físicas de la implementación de tal manera que puedan ser mejor entendidas y manejadas. Este modelo define la manera en que los diferentes módulos son organizados para poder controlar sus versiones, eliminarlos o distribuirlos. Además logra organizar las personas o equipos de trabajo.

La aplicación consta de una gran cantidad de código que es de soporte. Como se dijo anteriormente, fue necesario realizar un modelo de delegados, código de soporte para la GUI de G3D así como diferentes utilidades generales que se tuvieron que construir. Es por esto que no es objetivo del presente trabajo de diploma mostrar en detalle toda esta implementación sino que solo nos concentraremos en abordar las fuentes más vinculadas a los casos de usos descritos en la fase de *Análisis y Diseño*.

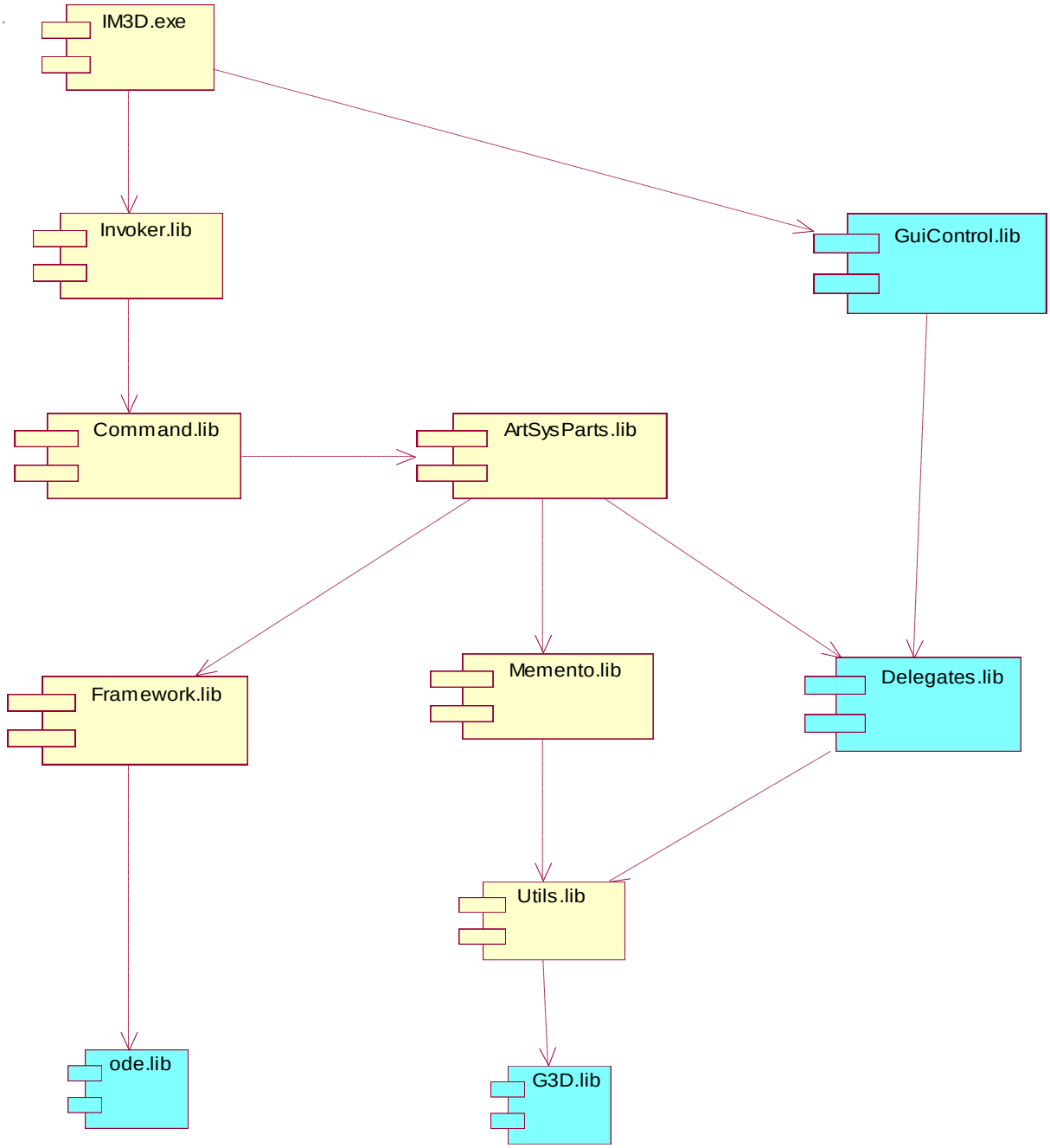
El producto final es una aplicación de escritorio, monolítica, instalada en una sola máquina. Su uso es exclusivamente productivo y solamente es utilizada por el desarrollador de los modelos físicos.

Los subsistemas que componen la aplicación son:

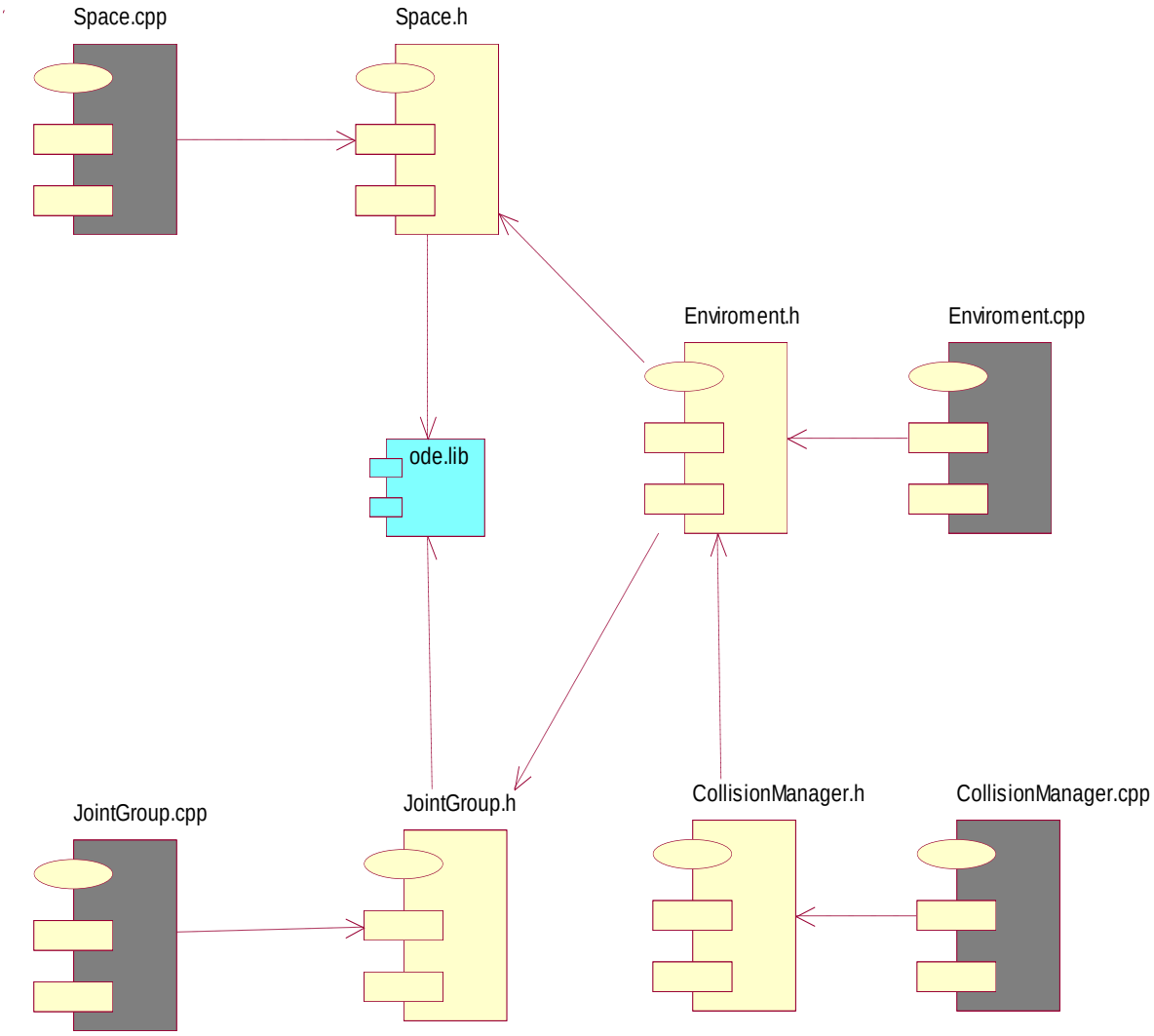
1- **Subsistema de simulación física ODE**. Este subsistema es la biblioteca ODE descrita anteriormente.
2- **Subsistema de gráficos G3D.** Este subsistema es la biblioteca G3D descrita anteriormente.
3- **Subsistema de útiles**. Aquí se encuentran los tipos y enumerados generales además de funciones útiles para convertir objetos a cadenas y rutinas para pintar algunas primitivas no contempladas en G3D.
4- **Subsistema de delegados**. Este subsistema es bastante pequeño y su empleo es exclusivamente para soportar el modelo de delegados.
5- **Subsistema GUI**. Este subsistema se encarga de controlar todos los componentes gráficos usados en la aplicación. Su descripción no es propósito de la tesis. Su utilización persigue como objetivo complementar y dar soporte a los controles GUI que presenta G3D en su versión 7.0, aún en desarrollo.

6- **Subsistema *Framework*.** Es el subsistema encargado de brindar las posibilidades de ODE a la aplicación. Este subsistema es el que se brinda a terceras personas para que pueda ser reutilizado el producto final.
7- **Subsistema Memento.** Este subsistema está pensado para almacenar información de los objetos, de manera que la misma pueda ser capturada y exteriorizada sin violar los principios de encapsulamiento, pudiéndose restaurar los mismos a sus estados previos.
8- **Subsistema ArtSysParts.** Este subsistema esta hecho para poder manipular información de los sistemas articulados, cuerpos rígidos, articulaciones y sus propiedades.
9- **Subsistema de Comandos.** Este subsistema separa la capa de presentación de la capa del negocio. Permite encapsular las solicitudes de los usuarios en objetos, posibilitando así deshacer o rehacer acciones. Se determinó hacer un subsistema debido a la gran cantidad de acciones que se podían tomar utilizando el sistema actualmente y a la posibilidad real de una extensión en este punto.
10- **Subsistema *Invoker*.** Para poder asociar la capa de presentación con los delegados y los comandos de manera eficiente se hizo necesario construir este subsistema. Permite un bajo acoplamiento al mantener desacopladas las capas de presentación y del negocio, haciendo muy fácil su reutilización.
11- **Subsistema IM3D.** Este subsistema es el corazón de la aplicación y es el encargado de inicializar la aplicación e integrar todos lo sistemas en él.

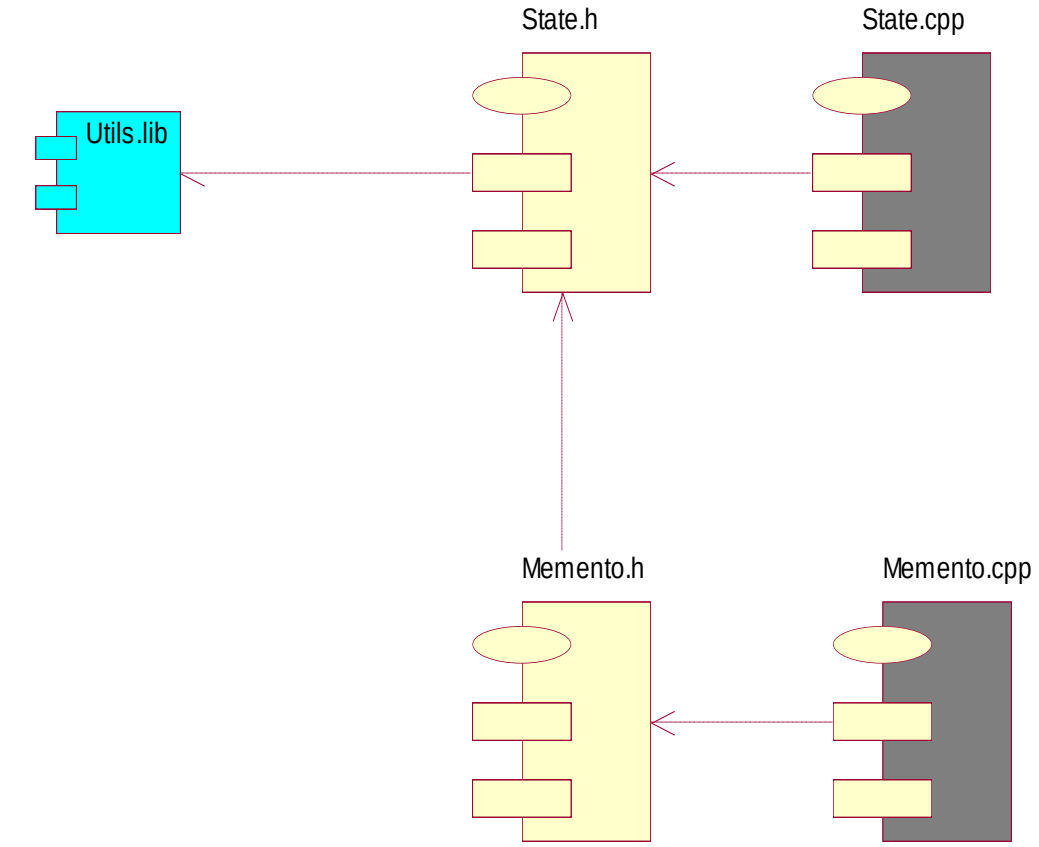
4.2 Diagrama de Subsistemas



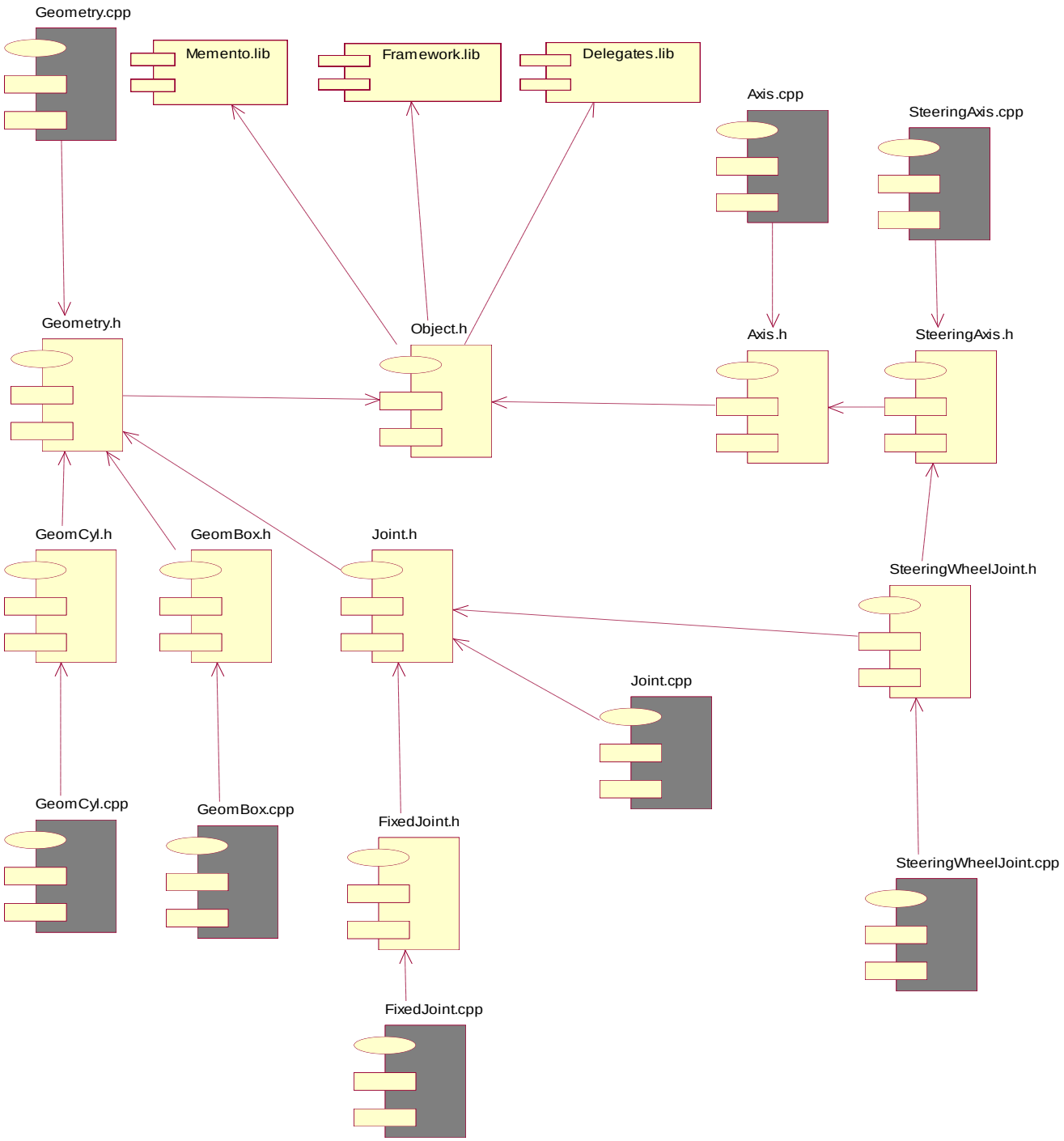
4.2.1 Subsistema Framework



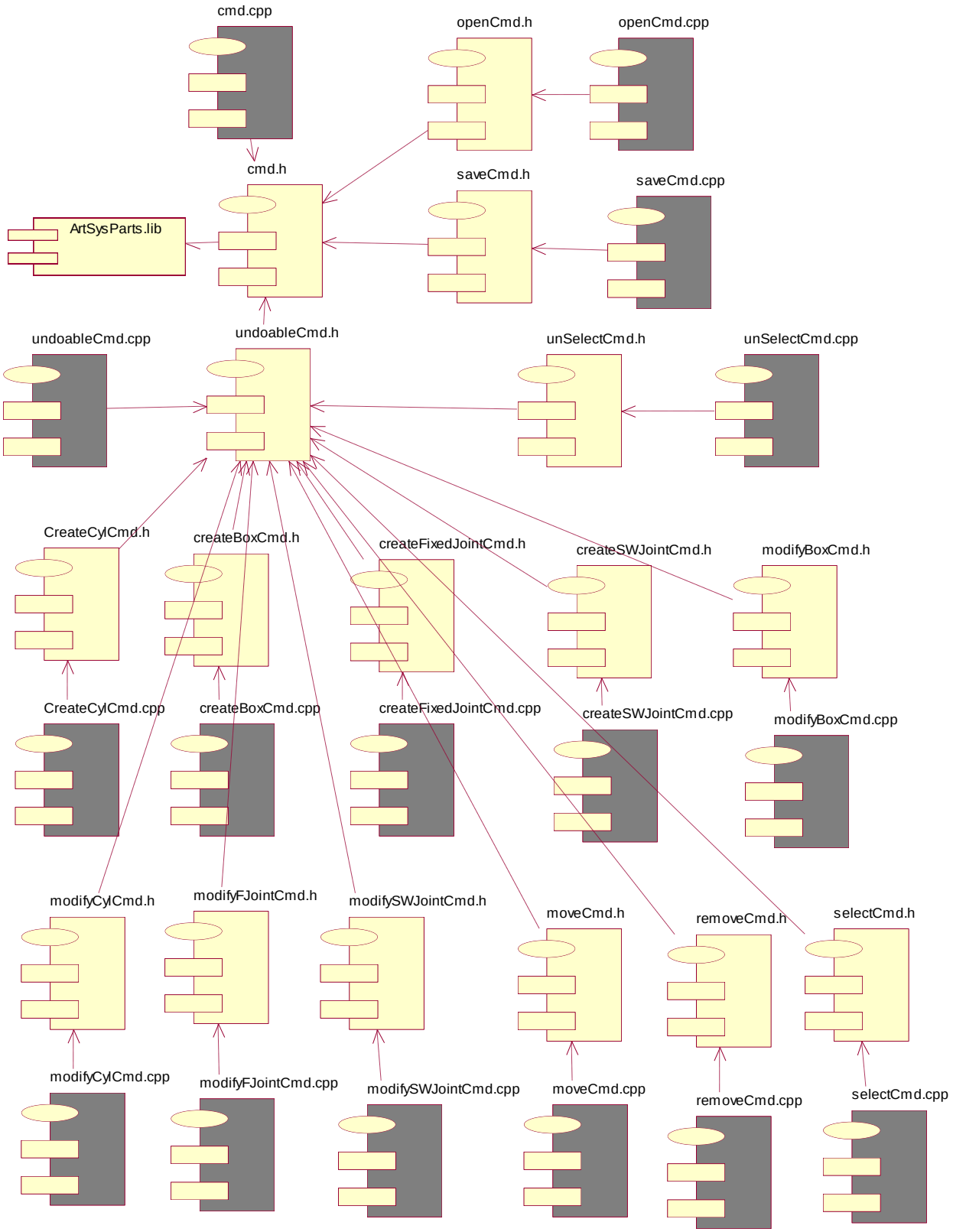
4.2.2 Subsistema Memento



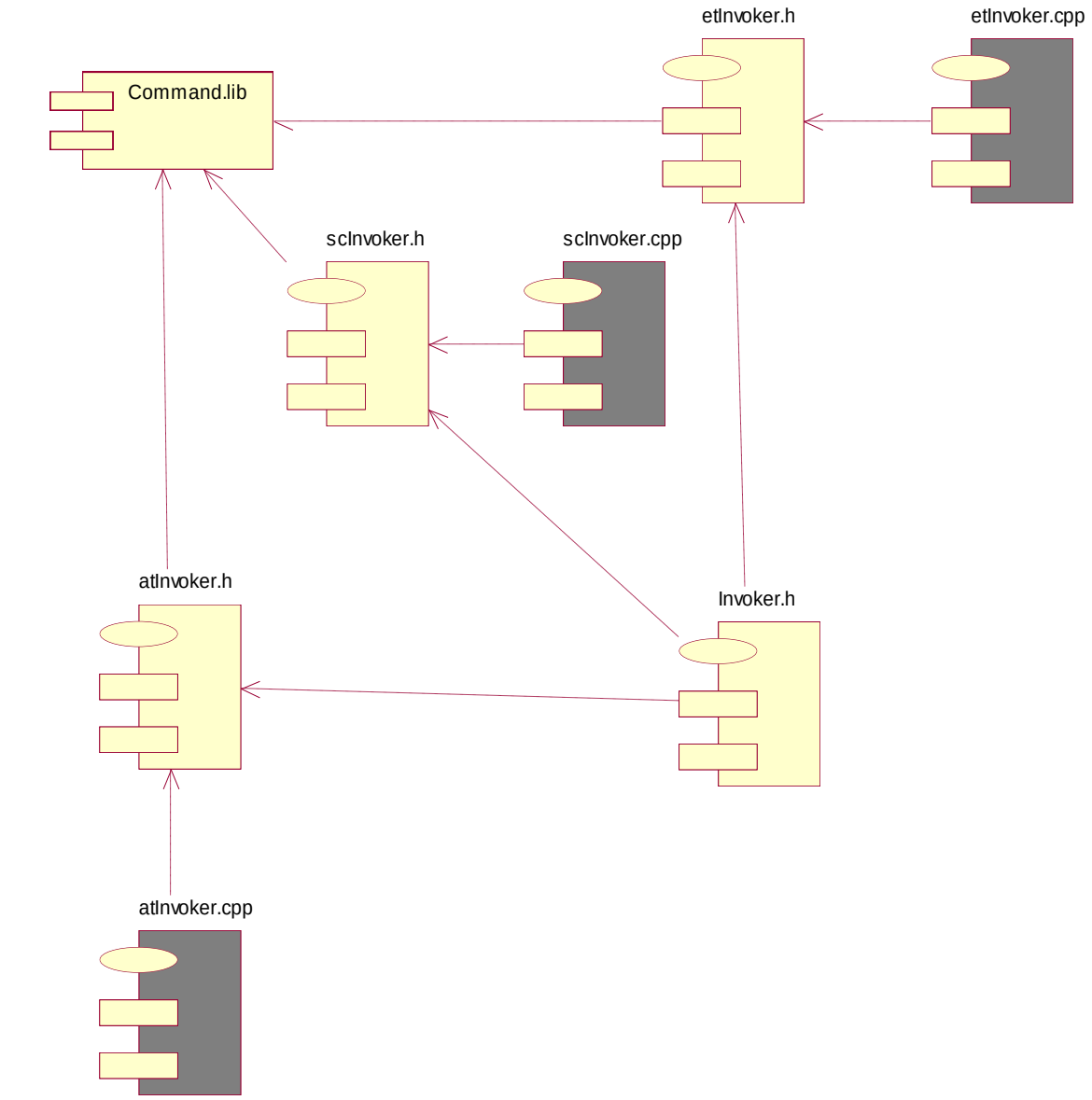
4.2.3 Subsistema ArtSysParts



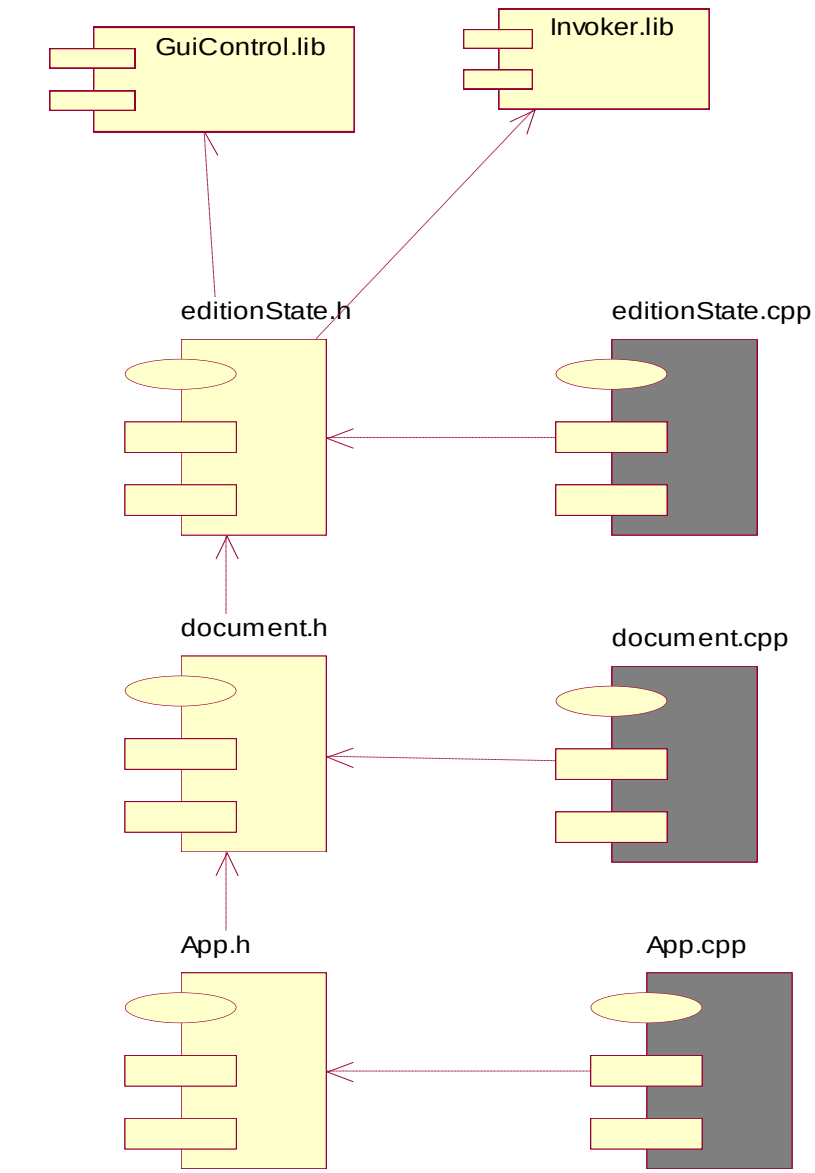
4.2.4 Subsistema de Comando



4.2.5 Subsistema Invoker



4.2.6 Subsistema IM3D



CAPÍTULO 4
IMPLEMENTACIÓN

4.3 Estándares de codificación

Los estándares de codificación son reglas específicas de un lenguaje, que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores. Los estándares de codificación no destapan problemas existentes, evitan más bien que los errores ocurran. Los errores frecuentes en programas pueden ser detectados mucho antes o pueden ser incluso evitados totalmente. Durante el desarrollo, los estándares de codificación ayudan a los ingenieros a producir un código de alta calidad y a entender y utilizar el código de sus colegas. Pero también realzan considerablemente la capacidad de mantenimiento y reutilización a largo plazo del producto final. Tal práctica del control de errores en el proceso del desarrollo mejora la calidad a la vez que reduce el tiempo de desarrollo, el costo y el esfuerzo.

4.3.1 Reglas de codificación

Constantes: Los nombres de constantes se escriben en mayúsculas. En caso de estar compuesta por más de una palabra, estas se separan por un carácter '_'.

Variables: Las variables se escriben con minúsculas; las variables con nombres compuestos, comienzan con la primera palabra en minúscula y el resto comenzando con letra inicial mayúscula.

Parámetros de métodos: Cumplen con la misma regla de las variables.

Clases: Los nombres de las clases se escriben con letra inicial mayúscula; en caso de nombre compuesto cada uno se escribe a continuación del otro y también con letra inicial mayúscula.

Métodos miembros de clases: Los métodos miembros de clases siguen la notación de las variables.

Nombres de enumerados: El nombre de los enumerados sigue la notación de las clases y sus valores la misma notación que las constantes.

Comentarios: Se usan los comentarios en línea para facilitar la comprensión del código, sobre todo en procedimientos complejos.

CONCLUSIONES

Con la realización de la presente investigación se arribaron a las siguientes conclusiones:

- La utilización de los sistemas articulados y del entorno gráfico para su edición, reduce de manera sustancial los conocimientos físicos para su creación y permite a la imaginación jugar un papel primario.
- El uso de la herramienta propuesta permite un mejor entendimiento de la biblioteca ODE, potenciando la exploración de muchas funcionalidades de manera sencilla, anteriormente desconocidas.
- Las utilidades del editor propuesto, relacionadas con el cambio de propiedades y relaciones entre los objetos, permite el logro de disímiles efectos durante las colisiones de forma parametrizable, el descubrimiento de nuevas relaciones posibles y un realismo mucho más perfeccionado.
- La inclusión de un módulo de simulación y un entorno de prueba en el editor propuesto garantizan la comodidad de los desarrolladores, al poder probar en la misma aplicación el sistema articulado creado sin preocuparse por la creación de un entorno confiable donde probar su modelo.
- La facilidad de uso y las potencialidades del editor propuesto garantizan un incremento de la productividad de los desarrolladores en términos de una disminución dramática del tiempo para producirlos, incluso de meses.
- El módulo con las clases necesarias para reutilizar el sistema articulado creado (*framework*) en la herramienta propuesta permite, con pocas instrucciones de código, cargar el modelo creado e insertarlo en cualquier entorno interactivo. Dicho *framework,* además brinda soporte a los procesos de simulación, si así se desea, mediante la abstracción de toda la manipulación de colisiones.
- Resulta ventajosa la utilización del editor propuesto en la modelación de sistemas articulados de cuerpos rígidos para las animaciones en los videojuegos.

Es necesario destacar que el proceso de desarrollo del sistema no supuso gastos de recursos ya que la infraestructura de producción estaba creada. Además, el soporte utilizado para la realización de la aplicación es totalmente libre y multiplataforma por lo que no se incurrieron en gastos referentes al pago de licencias y se siguieron las políticas de *software* libre.

RECOMENDACIONES

A lo largo del desarrollo del presente trabajo, han ido surgiendo ideas que podrían implementarse en un futuro, de forma que se logre una aplicación más útil y efectiva, para lo cual se recomienda:

1- Añadir otras primitivas básicas contempladas en la biblioteca ODE, con el objetivo de lograr una mejor aproximación durante la descomposición del modelo visual en primitivas (esferas, cápsulas, planos).

2- Añadir otros tipos de articulaciones contempladas en la biblioteca ODE, con el objetivo de adicionar otros efectos y animaciones (bisagra, universal, pistón, rótula).

3- Aprovechando las potencialidades de la biblioteca G3D relacionadas con la carga de modelos 3D, efectuar la carga del modelo visual en cuestión, con el objetivo de integrar en una sola herramienta el proceso de descomposición.

4- Proveer una interfaz para alinear los cuerpos en la escena.

5- Dar continuidad al ciclo de desarrollo número dos.

BIBLIOGRAFÍA

1. **Rahnejat, Homer and Rothberg, Steve.** *Multi-Body Dynamics: Monitoring and Simulation Techniques III.* Loughborough : John Wiley and Sons, 2004. p. 544. 1860584632.

2. **Coutinho, Murilo Gondim.** *Dynamic Simulation of Multibody System.* New York : Springer-Verlag, 2001.

3. *Particle systems. A technique for modeling a class of fuzzy objects.* **Reeves, W.T.** s.l. : *Proc. ACM Transactions on Graphics*, Abril 1983, Vol. 2, pp. 91-108.

4. *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems.* **Reeves, W.T. and R., Blau.** San Francisco, California : Proc. SIGGRAPH, Julio 1985.

5. **Smith, Alvy Ray.** *The adventure of Andre & Wally B.* Pixar, 1984.

6. *Modeling Waves and Surf.* **Peachey, Darwyn R.** s.l. : *ACM Computer Graphics*, 1986, Vol. 20.

7. *A Simple Model of Ocean Waves.* **Fournier, Alain.** s.l. : *ACM Computer Graphics*, 1989, Vol. 20.

8. *Flocks, Herds, and Schools: A Distributed Behavioral Model.* **Reynolds, Craig.** s.l. : *ACM Computer Graphics*, 1987, Vol. 21.

9. *Application of a General Particle System Model to Movement of Pedestrians and Vehicles.* **Schaefer, Lisa.** s.l. : ACM SIGRAPH, 1998.

10. *Realtime Simulation of Dust Behavior Generated by a Fast Moving Vehicle.* **Chen, Jim X.** s.l. : *ACM Modeling and Computer Simulation*, Abril 1999, Vol. 9.

11. *Rendering Fur in 3 Dimensions.* **Kajira, James T.** s.l. : ACM Computer Graphics, Julio 1989, Vol. 23.

12. *Surface Modeling with Orientated Particle Systems.* **Szeliski, Richard.** s.l. : *ACM Computer Graphics*, Julio 1992, Vol. 26.

13. *Isosurface Extraction using Particle Systems.* **Crossno, Patricia.** s.l. : *ACM Computer Graphics*, Julio 1992, Vol. 26.

14. *Interactive Modeling Using Particle Systems.* **Leech, J. P. and R. M. Taylor.** MIT : *Proc. of 2nd Conference on Discrete Element*, 1993.

15. *Flow and Change in Appearance.* **Dorsey, Julie.** s.l. : *ACM Computer Graphics*, 1996.

16. *Particle Animation and Rendering Using Data Parallel Computation.* **Sims, Karl.** s.l. : *ACM Computer Graphics*, Agosto 1990, Vol. 24.

17. *Practical Animation of Liquids.* **Foster, Nick.** s.l. : *ACM Computer Graphics*, Agosto 2001.

18. *The Design of an API for Particle Systems.* **McAllister, David K.** *University of North Carolina at Chapel Hill : Department of Computer Science*, 2000.

19. *Computational Modeling for the Computer Animation.* **Girard, Michael and Maciejewski,.** s.l. : *Computer Graphics*, Julio 1985, Vol. 19, pp. 263-260.

20. *A Kinematic Model of the Human Spine and Torso.* **Monheit, G., Badler.** s.l. : *IEEE Comput. Graph.*, Marzo 1991, Vol. 11, pp. 29-38.

21. *Morphology-independent representation of motions for interactive human-like animation.* **Kulpa. R., Multon F., Arnaldi B.** s.l. : Computer Graphics Forum, 2005, Vol. 24, pp. 343-352.

22. *Anatomy-based joint models for virtual human skeletons.* **Maciel, A., Nedel, L.P. and Dal Sasso Freitas, C.M.** s.l. : *Proc. Computer Animation,* 2002, pp. 220-224.

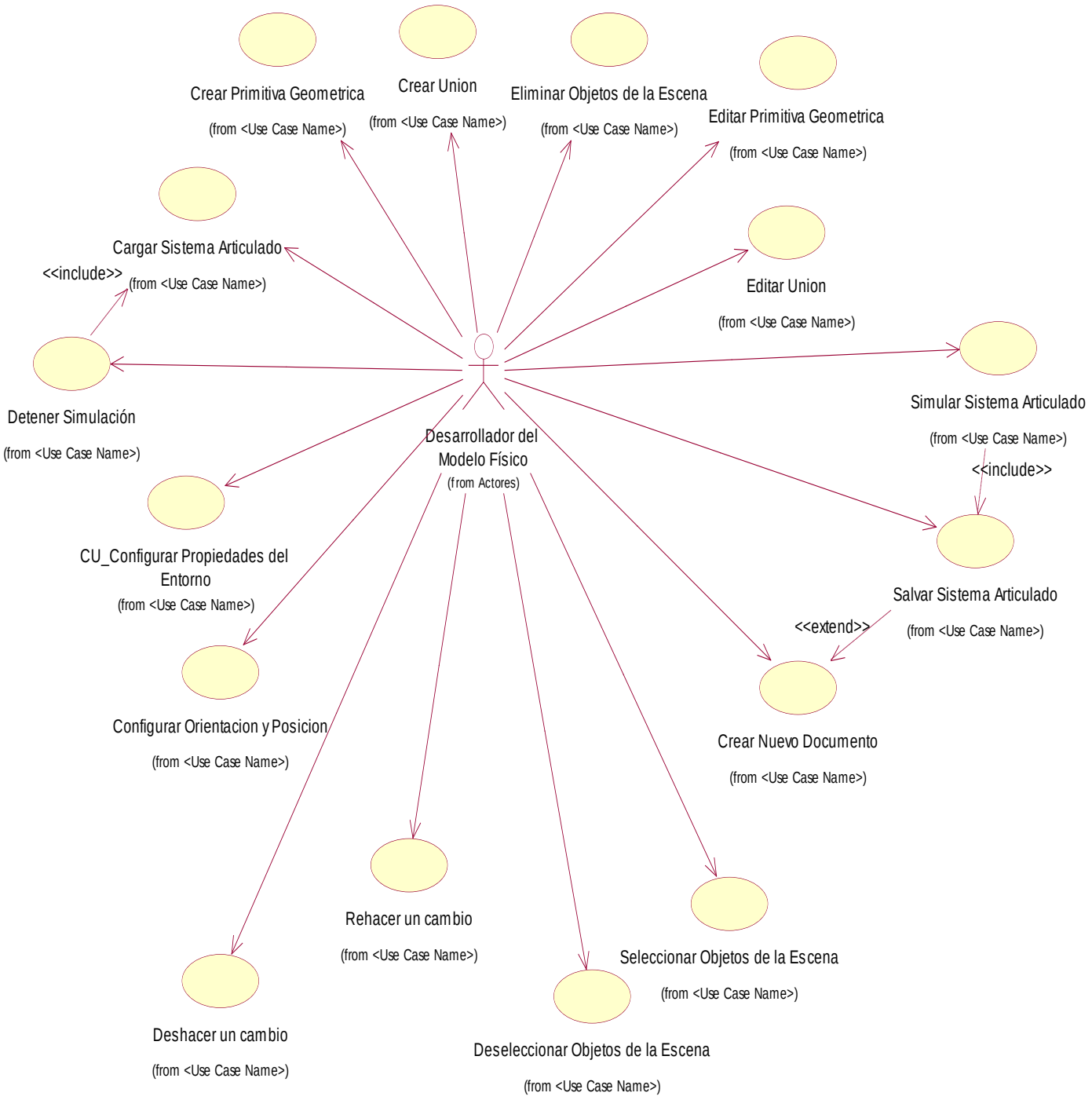
23. *Hierarchical implicit surface joint limits for human body tracking.* **Herda, L., Urtasun, R., Fua, P.** s.l. : *Computer Vision and Image Understanding,* 2005, Vol. 99, pp. 189-209.

24. *EXPLOITING COUPLED JOINTS. Anatomic Control of the Spine with IK Through Linearly Coupled Joints.* **Raunhardt, Daniel and Boulic, Ronan.** s.l. : *VRLAB, Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland,* 2007.

25. *A Geometric Investigation of Reach.* **Korein, J. U.** s.l. : MIT Press., 1985.
26. *Human shoulder modeling including scapulo-thoracic constraint and joint sinus cones.* **Maurel, W., Thalmann, D.** s.l. : *Computers and Graphics,* 2000, Vol. 24, pp. 203-218.
27. *A general joint component framework for realistic articulation in human characters.* **Shao, W., Ng-Thow-Hing, V.** Monterrey, California : *Symposium on interactive 3D Graphics*, Abril 27-30, 2003.
28. **Shabana, A.** *Computational Dynamics .* New York : John Wiley and Sons Inc, 2001.
29. *Forward dynamics based realistic animation of rigid bodies.* **Park, Jihun and Fussell, Donald S.** 4, s.l. : *Elsevier Science Ltd.* , Julio-Agosto 1997, Vol. 21, pp. 483-496 .
30. *Impulse-based dynamic simulation.* **Bender, Jan.** *Karlsruhe, Alemania* : s.n., 2007.
31. **Nelson.** *The BSD License*. *Open Source Initiative.* [*Online*] Octubre Martes, 31, 2006. http://www.opensource.org/licenses/bsd-license.php.
32. **McGuire, Morgan.** *G3D Engine ScreenShots*. *G3D Engine.* [*Online*] 2007. http://g3d-cpp.sourceforge.net/screenshots.html.
33. **Smith, Russel.** *ODE Documentation. Open Dynamics Engine.* [*Online*] Mayo 18, 2007. http://www.ode.org.
34. **Álvarez, Sofía and Hernández, Anaisa.** Metodología para el desarrollo de aplicaciones con tecnología Orientada a Objetos utilizando notación UML. La Habana : s.n., 2000.
35. **Booch, Grady, Jacobson, Ivar and Rumbaugh, James.** El Proceso Unificado de Desarrollo de Software. Madrid : Addison Wesley, 2000. 8478290362.
36. **Corporation, Microsoft.** *Event Handlig in Native C++*. *MSDN Library for Visual Studio 2005.* 2005.
37. **Corporation, Borland.** *Keyword extensions. Borland C++ Builder Help.* 2002.
38. **McGuire, Morgan.** *G3D Documentation. G3D Engine.* [*Online*] 2007. http://g3d-cpp.sourceforge.net/manual/index.html.
39. **Ibarra Nuño, César and Recinos Silva, Mario Alberto.** Algoritmo de *Denavit- Hartenberg* para la obtención del modelo. [*Online*] 2006 http://148.202.12.20/~cin/robotic/tarease/dh/dh.htm.

ANEXOS

Anexo 1. Diagrama de casos de uso del sistema



Anexo 2. Casos de uso por ciclo

Código	Nombre del caso de uso	Paquete	Justificación de la selección
1	Crear un Nuevo Documento.		Permite la creación de un documento en blanco para la construcción de un nuevo sistema articulado.
	Crear Primitiva Geométrica.		Permite la creación de las primitivas geométricas en el entorno de modelación.
	Seleccionar Objetos en la Escena.		Permite la selección de primitivas o uniones, sobre las cuales se realizara una acción posteriormente.
	Deseleccionar Objetos de la Escena.		Permite la deselección de objetos de la escena, descartando toda acción sobre estos.
	Eliminar Objetos de la Escena.		Permite la eliminación de una primitiva una vez que ya no sea necesaria.
	Editar Primitiva Geométrica.		Permite la edición de las propiedades tanto físicas como graficas de una primitiva seleccionada.
	Configurar Orientación y Posición.		Permite el cambio de posición y orientación de una primitiva seleccionada.
	Deshacer un cambio.		Permite la recuperación ante errores o situaciones tentadoras.
	Rehacer un cambio.		Permite rehacer inmediatamente un cambio deshecho.
2	Crear Unión.		Permite la creación de una unión entre dos geometrías seleccionadas.
	Editar Unión.		Permite cambiar la configuración de la unión, tanto sus propiedades físicas como gráficas.
3	Configurar Parámetros Físicos Globales.		Permite establecer las propiedades del mundo virtual en el que se probara el sistema articulado.
	Simular Sistema Articulado.		Permite probar el sistema articulado en un entorno cargado de disimiles eventos.
	Detener Simulación.		Permite detener la simulación en curso, situando el modelo en su posición inicial.
	Salvar Sistema Articulado.		Permite salvar el sistema articulado a un fichero binario, para que pueda ser reutilizado por terceros.
	Cargar Sistema Articulado.		Permite la carga desde fichero de un sistema articulado para su posterior edición.

Anexo 3. Casos de uso expandidos

CU-1	**Crear Nuevo Documento**
Propósito	Crear un documento en blanco.
Actores	Diseñador del Modelo Físico.
Resumen: El Diseñador del Modelo Físico crea un documento en blanco.	
Referencias:	RF: 1
Precondiciones	El diseñador seleccionó la opción crear nuevo documento.
Acción del autor	**Respuesta del sistema**
1) El Diseñador del Modelo Físico escoge la opción de crear un nuevo documento.	2) El sistema chequea la existencia de algún archivo abierto sin salvar. 3) En caso negativo cierra el documento actual y abre uno nuevo.
Flujo Alterno 1	
Acción del autor	**Respuesta del sistema**
	3) En caso positivo muestra un cuadro de diálogo solicitando confirmación para salvar el documento actual.
4) El desarrollador del modelo físico confirma.	5) El sistema salva el documento actual. 6) Cierra el documento actual y abre uno nuevo.
Flujo Alterno 2	
Acción del autor	**Respuesta del sistema**
4) El desarrollador del modelo físico selecciona no guardar el modelo.	5) Cierra el documento actual y abre uno nuevo.
Puntos de Extensión: **Línea 5:** Ver CU-13 Salvar Sistema Articulado.	
Poscondiciones	Se abre un nuevo documento.

CU-2	**Crear Primitiva Geométrica**
Propósito	Crear una primitiva geométrica e introducirla en el entorno de modelación.
Actores	Diseñador del Modelo Físico.
Resumen: El Diseñador del Modelo Físico crea una primitiva geométrica llenando un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 2
Precondiciones	El desarrollador selecciona crear un tipo de primitiva geométrica.
Acción del autor	**Respuesta del sistema**
1) El desarrollador del modelo físico escoge un tipo de geometría para crearla.	
Flujo Alterno 1	
1) El desarrollador del modelo físico escoge la opción de crear una caja.	2) Ver Sección Crear Caja.
Flujo Alterno 2	
1) El desarrollador del modelo físico escoge la opción de crear un cilindro.	2) Ver Sección Crear Cilindro.

Sección	**Crear Caja**
CU al que pertenece	Crear Primitiva Geométrica.
Propósito	Crear una caja e introducirla en el entorno de modelación.
Actores	Desarrollador del Modelo Físico.
Resumen: El Diseñador del Modelo Físico crea una caja llenando un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 2.1
Precondiciones	El diseñador selecciona la opción crear una nueva caja.

Acción del autor	**Respuesta del sistema**
1) El desarrollador del modelo físico escoge la opción de crear una caja.	2) El sistema muestra un formulario con las entradas necesarias para la creación de la caja.
3) El desarrollador del modelo físico llena el formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando crear caja y se archiva el comando en el historial. 6) El sistema muestra en el entorno de modelación la caja con las propiedades especificadas por el usuario. 7) El sistema limpia el formulario pudiéndose crear una caja nueva.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión:	
Poscondiciones:	Se introduce una nueva caja en el entorno de modelación 3D.

Sección	**Crear Cilindro**
CU al que pertenece	Crear Primitiva Geométrica.
Propósito	Crear un cilindro e introducirlo en el entorno de modelación.
Actores	Desarrollador del Modelo Físico.
Resumen: El Desarrollador del Modelo Físico crea un cilindro llenando un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 2.2
Precondiciones	El diseñador selecciona la opción crear un nuevo cilindro.

Acción del autor	**Respuesta del sistema**
1) El desarrollador del modelo físico escoge la opción de crear un cilindro.	2) El sistema muestra un formulario con las entradas necesarias para la creación del cilindro.
3) El desarrollador del modelo físico llena el formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando crear cilindro y se archiva el comando en el historial. 6) El sistema muestra en el entorno de modelación el cilindro con las propiedades especificadas por el usuario. 7) El sistema limpia el formulario pudiéndose crear un cilindro nuevo.

Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión:	
Poscondiciones:	Se introduce un nuevo cilindro en el entorno de modelación 3D.

CU-3	**Crear Unión**
Propósito	Crear una unión entre dos geometrías previamente seleccionadas.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a crear una unión entre dos geometrías, llenando un formulario con propiedades para la unión, mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 3
Precondiciones:	El diseñador selecciona las dos geometrías.
Acción del autor	**Respuesta del sistema**
1) El Diseñador del Modelo Físico selecciona dos geometrías.	2) El sistema habilita las opciones para la creación de las distintas uniones.
Flujo Alterno 1	
3) El diseñador del Modelo Físico selecciona crear una unión fija.	4) Ver sección Crear Unión Fija.
Flujo Alterno 2	
3) El diseñador del Modelo Físico selecciona crear una unión con suspensión.	4) Ver sección Crear Unión con Suspensión.
Sección	**Crear Unión Fija**
CU al que pertenece	Crear Unión.
Propósito	Crear una unión fija entre dos geometrías previamente seleccionadas.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a crear una unión fija entre dos geometrías, llenando un formulario con propiedades para la unión, mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 3.1
Precondiciones:	El diseñador selecciona las dos geometrías.
Acción del autor	**Respuesta del sistema**
3) El diseñador del Modelo Físico selecciona crear una unión fija.	4) El sistema muestra un formulario con las entradas necesarias para la creación de una unión fija.
5) El diseñador del modelo físico llena el formulario y da clic en aceptar.	6) El sistema verifica la validez de los datos. 7) En caso afirmativo se ejecuta el comando crear unión fija y se archiva dicho comando en el historial. 8) El sistema muestra en el entorno de modelación la unión fija entre las geometrías previamente seleccionadas por el usuario. 9) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**

	7) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión:	
Poscondiciones:	Se crea una unión fija entre las dos geometrías seleccionadas.
Sección	**Crear Unión con Suspensión**
CU al que pertenece	Crear Unión.
Propósito	Crear una unión con suspensión entre dos geometrías previamente seleccionadas.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a crear una unión con suspensión entre dos geometrías, llenando un formulario con propiedades para la unión, mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 3.2
Precondiciones:	El diseñador selecciona las dos geometrías.
Acción del autor	**Respuesta del sistema**
1) El diseñador del Modelo Físico selecciona crear una unión con suspensión.	2) El sistema muestra un formulario con las entradas necesarias para la creación de una unión con suspensión.
3) El diseñador del modelo físico llena el formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando crear unión con suspensión y se archiva dicho comando en el historial. 6) El sistema muestra en el entorno de modelación la unión con suspensión entre las geometrías previamente seleccionadas por el usuario. 7) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	7) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión:	
Poscondiciones:	Se crea una unión con suspensión entre las dos geometrías seleccionadas.

CU-4	**Seleccionar Objetos de la Escena**
Propósito	Seleccionar uno o varios objetos de la escena.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del Modelo Físico selecciona tanto geometrías como uniones en la escena.	
Referencias:	RF: 4, 5
Precondiciones:	El objeto debe existir en la escena.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico presiona la tecla *control* y selecciona con clic un objeto de la escena.	2) El sistema resalta el objeto seleccionado. 3) El sistema lo adiciona a la lista de elementos seleccionados.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión:	

Poscondiciones:	El objeto queda seleccionado.
CU-5	**Deseleccionar Objetos de la Escena**
Propósito	Deseleccionar uno o varios objetos de la escena.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del Modelo Físico deselecciona tanto geometrías como uniones en la escena.	
Referencias:	RF: 6, 7
Precondiciones:	El objeto debe estar seleccionado en la escena.

Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico presiona la tecla *control* y da clic en un objeto seleccionado de la escena.	2) El sistema deselecciona el objeto dejando de resaltarlo. 3) El sistema lo elimina a la lista de elementos seleccionados.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión:	
Poscondiciones:	El objeto queda deseleccionado.

CU-6	**Eliminar objetos de la escena**
Propósito	Eliminar uno o varios objetos de la escena.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico elimina tanto una o varias geometrías o uniones.	
Referencias:	RF: 8, 9
Precondiciones	Los objetos deben estar seleccionados.

Acción del autor	**Respuesta del sistema**
1) El Diseñador del Modelo Físico selecciona uno o varios objetos de la escena. 2) El Diseñador del Modelo Físico oprime la tecla *delete*.	3) El sistema ejecuta el comando eliminar objeto de la escena y archiva el estado anterior en el historial.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión	
Poscondiciones:	Los objetos seleccionados son eliminados de la escena.

CU-7	**Editar Primitiva Geométrica**
Propósito	Editar las propiedades de una primitiva geométrica.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a editar las propiedades de una primitiva geométrica a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 10, 10.1, 10.2
Precondiciones:	El diseñador selecciona con clic derecho una primitiva.

Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic.	

derecho sobre una primitiva en la escena.	
Flujo Alterno 1	
1) El diseñador del modelo físico da clic derecho sobre una caja.	2) Ver sección Editar Caja.
Flujos Alternos 2	
1) El diseñador del modelo físico da clic derecho sobre un cilindro.	2) Ver sección Editar Cilindro.

Sección	**Editar Caja**
CU al que pertenece	Editar Primitiva Geométrica.
Propósito	Editar las propiedades de una caja.
Actores	Desarrollador del modelo físico.
Resumen: El desarrollador del modelo físico se dispone a editar las propiedades de una caja a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 10.1
Precondiciones:	El diseñador selecciona con clic derecho una caja.

Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic derecho sobre una caja en la escena.	2) El sistema muestra un formulario con las propiedades actuales de dicha caja.
3) El diseñador del modelo físico edita las propiedades deseadas a través del formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando editar caja y se archiva el estado anterior en el historial. 6) El sistema muestra en el entorno de modelación la caja con las propiedades especificadas por el usuario. 7) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	

Poscondiciones:	Las propiedades de la caja son editadas.
Sección	**Editar Cilindro**
CU al que pertenece	Editar Primitiva Geométrica.
Propósito	Editar las propiedades de un cilindro.
Actores	Desarrollador del modelo físico.
Resumen: El desarrollador del modelo físico se dispone a editar las propiedades de un cilindro a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 10.2
Precondiciones:	El diseñador selecciona con clic derecho un cilindro.

Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic derecho sobre un cilindro en la escena.	2) El sistema muestra un formulario con las propiedades actuales de dicho cilindro.
3) El diseñador del modelo físico edita las propiedades deseadas a través del formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando editar cilindro y se archiva dicho comando en el historial.

	6) El sistema muestra en el entorno de modelación el cilindro con las propiedades especificadas por el usuario. 7) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Las propiedades del cilindro son editadas.

CU-8	**Editar Unión**
Propósito	Editar las propiedades de una unión.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a editar las propiedades de una unión a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 11, 11.1, 11.2
Precondiciones:	El diseñador selecciona con clic derecho una unión.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic derecho sobre una unión en la escena.	
Flujo Alterno 1	
1) El diseñador del modelo físico da clic derecho sobre una unión fija.	2) Ver sección Editar Unión Fija.
Flujos Alternos 2	
1) El diseñador del modelo físico da clic derecho sobre una unión con suspensión.	2) Ver sección Editar Unión con Suspensión.
Sección	**Editar Unión Fija**
CU al que pertenece	Editar Unión.
Propósito	Editar las propiedades de una unión fija.
Actores	Desarrollador del modelo físico.
Resumen: El desarrollador del modelo físico se dispone a editar las propiedades de una unión fija a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 11.1
Precondiciones:	El diseñador selecciona con clic derecho una unión fija.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic derecho sobre una unión fija.	2) El sistema muestra un formulario con las propiedades actuales de dicha unión fija.
3) El diseñador del modelo físico edita las propiedades deseadas a través del formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando editar unión fija y se archiva dicho comando en el historial. 6) El sistema muestra en el entorno de modelación la unión fija con las propiedades especificadas por el usuario. 7) El sistema cierra el formulario.

Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Las propiedades de la unión fija son editadas.
Sección	**Editar Unión con Suspensión**
CU al que pertenece	Editar Unión.
Propósito	Editar las propiedades de una unión con suspensión.
Actores	Desarrollador del modelo físico.
Resumen: El desarrollador del modelo físico se dispone a editar las propiedades de una unión con suspensión a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 11.2
Precondiciones:	El diseñador selecciona con clic derecho una unión con suspensión.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic derecho sobre una unión con suspensión en la escena.	2) El sistema muestra un formulario con las propiedades actuales de dicha unión con suspensión.
3) El diseñador del modelo físico edita las propiedades deseadas a través del formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando editar unión con suspensión y se archiva dicho comando en el historial. 6) El sistema muestra en el entorno de modelación la unión con suspensión con las propiedades especificadas por el usuario. 7) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Las propiedades de la unión con suspensión son editadas.

CU-9	**Configurar Orientación y Posición**
Propósito	Configurar la orientación y la posición en tiempo real de una o varias geometrías seleccionadas en la escena.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del Modelo Físico se dispone a configurar en tiempo real la orientación y la posición de una o varias geometrías en la escena a través de un manipulador amigable mostrado como respuesta a la petición.	
Referencias:	RF: 12
Precondiciones:	Las geometrías a configurar están seleccionadas.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico selecciona	2) El sistema muestra un manipulador con una estructura

una o varias geometrías en la escena.	preparada para cada tipo de movimiento de rotación o traslación.
3) El Diseñador del Modelo Físico pasa el mouse por encima del eje de traslación o el plano de rotación del manipulador sobre el cual desea hacer el movimiento respectivo.	4) El sistema resalta el tipo de movimiento.
5) El usuario da un clic sostenido y desplaza el mouse hacia la dirección deseada.	6) El sistema ejecuta el comando mover primitivas y archiva el estado anterior en el historial. 7) El sistema muestra en la escena las primitivas seleccionadas con la nueva orientación y posición.
8) El usuario libera el clic.	
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión	
Poscondiciones:	Las primitivas seleccionadas toman la nueva orientación y posición.

CU-10	**Configurar Propiedades del Entorno**
Propósito	Configurar las propiedades físicas del mundo virtual.
Actores	Diseñador del modelo físico.
Resumen: El Diseñador del modelo físico se dispone a configurar las propiedades físicas del mundo virtual a través de un formulario mostrado por el sistema como respuesta a la petición.	
Referencias:	RF: 13
Precondiciones:	El diseñador selecciona con un clic la opción de configurar los parámetros globales.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic sobre la opción de configuración de parámetros físicos globales.	2) El sistema muestra un formulario con las propiedades actuales del mundo virtual.
3) El diseñador del modelo físico edita las propiedades deseadas a través del formulario y da clic en aceptar.	4) El sistema verifica la validez de los datos. 5) En caso afirmativo se ejecuta el comando configurar propiedades globales y se archiva el estado anterior en el historial. 6) El sistema cierra el formulario.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	5) En caso de datos inválidos el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Los parámetros físicos globales son editados.

CU-11	**Simular Sistema Articulado**
Propósito	Probar el sistema articulado creado en un mundo virtual cargado de disímiles eventos.
Actores	Diseñador del Modelo Físico.
Resumen: El Diseñador del Modelo Físico se dispone a probar el sistema articulado creado en el mundo virtual.	

Podrá interactuar con este a través de teclas predeterminadas así como escoger diferentes vistas.	
Referencias:	RF: 14, 15, 16
Precondiciones:	El diseñador selecciona con un clic la opción de simular el sistema articulado.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico da clic sobre la opción de simular el sistema articulado.	2) El sistema muestra una ventana de información con las teclas predeterminadas para cambiar las diferentes vistas de simulación así como la opción de activación de las teclas para controlar el sistema articulado.
3) El Diseñador del Modelo Físico escoge activar las teclas de control y da clic en aceptar.	4) El sistema guarda en un fichero con el nombre previamente definido por el usuario o en su ausencia uno por defecto, el sistema articulado actual. 5) La aplicación simula el sistema articulado bajo la acción de la configuración global previamente establecida por el diseñador, o en su ausencia la configuración por defecto, en la vista actual y respondiendo a los eventos de teclado.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
3) El Diseñador del Modelo Físico no activa las teclas de control y da clic en aceptar.	4) El sistema guarda en un fichero con el nombre previamente definido por el usuario o en su ausencia uno por defecto, el sistema articulado actual. 5) La aplicación simula el sistema articulado bajo la acción de la configuración global previamente establecida por el diseñador, o en su ausencia la configuración por defecto, en la vista actual pero no responde a los eventos de teclado.
Puntos de Extensión: Línea 4: Ver CU-13 Salvar Sistema Articulado.	
Poscondiciones:	Se prueba el sistema articulado.

CU-12	**Cargar Sistema Articulado**
Propósito	Cargar un sistema articulado desde un fichero binario.
Actores	Diseñador del modelo físico.
Resumen: El diseñador del modelo físico se dispone a cargar un sistema articulado desde un fichero binario introduciéndolo en el entorno de modelación, a través de una interfaz que permitirá especificar la ruta del mismo en respuesta a la petición.	
Referencias:	RF: 17
Precondiciones:	El diseñador selecciona con un clic la opción de cargar un sistema articulado.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico escoge la opción de cargar un sistema articulado.	2) El sistema muestra una interfaz para especificar la ruta del archivo.
3) El diseñador del modelo físico especifica la ruta del fichero y da clic en aceptar.	4) El sistema verifica la validez de la ruta. 5) En caso afirmativo se ejecuta el comando cargar el archivo especificado. 6) El sistema cierra la interfaz y muestra en el entorno de modelación el sistema articulado cargado.

Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	2) En caso de que la ruta sea inválida el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Se carga el sistema articulado en el entorno de modelación.

CU-13	**Salvar Sistema Articulado**
Propósito	Salvar un sistema articulado para un fichero binario.
Actores	Diseñador del modelo físico.
Resumen: El diseñador del modelo físico se dispone a salvar un sistema articulado en un fichero binario, a través de una interfaz que permitirá especificar la ruta y el nombre del mismo en respuesta a la petición.	
Referencias:	RF: 18
Precondiciones:	El diseñador selecciona con un clic la opción de salvar un sistema articulado.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico escoge la opción de salvar un sistema articulado.	2) El sistema muestra una interfaz para especificar la ruta y el nombre del archivo.
3) El diseñador del modelo físico especifica los datos y da clic en aceptar.	4) El sistema verifica la validez de la ruta. 5) En caso afirmativo se ejecuta el comando salvar sistema articulado para la ruta especificada. 6) El sistema cierra la interfaz.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
	4) En caso de que la ruta sea inválida el sistema muestra un mensaje especificando el error correspondiente.
Puntos de Extensión	
Poscondiciones:	Se salva el sistema articulado en el fichero especificado.

CU-14	**Deshacer Un Cambio**
Propósito	Deshacer un cambio realizado.
Actores	Diseñador del modelo físico.
Resumen: El diseñador del modelo físico deshace un cambio realizado.	
Referencias:	RF: 19
Precondiciones:	El diseñador selecciona con un clic la opción de deshacer o presiona la combinación *control + z*.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico selecciona con un clic la opción de deshacer o presiona la combinación *control + z.*	2) El sistema busca en el historial el ultimo comando realizado y ejecuta su opción deshacer. 3) El sistema muestra íntegramente el entorno de modelación en su estado anterior.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión	

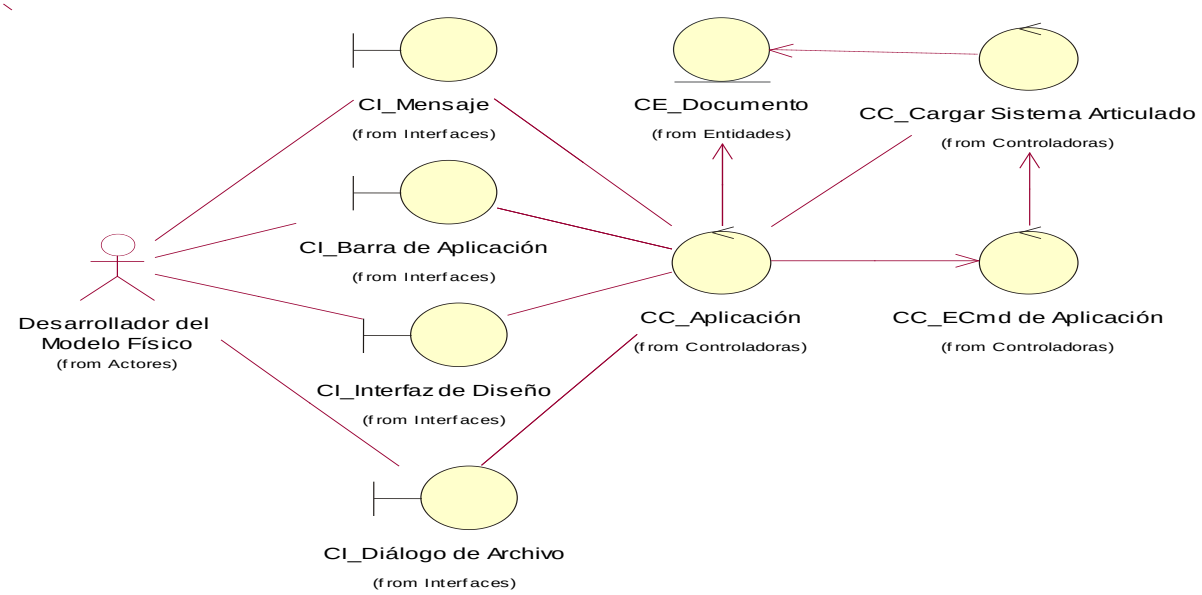
Poscondiciones:	El entorno de modelación vuelve a su estado anterior.

CU-15	**Rehacer Un Cambio**
Propósito	Rehacer un cambio realizado.
Actores	Diseñador del modelo físico.
Resumen: El diseñador del modelo físico rehace un cambio deshecho.	
Referencias:	RF: 20
Precondiciones:	El diseñador selecciona con un clic la opción de rehacer o presiona la combinación *control + y*.
Acción del autor	**Respuesta del sistema**
1) El diseñador del modelo físico selecciona con un clic la opción de rehacer o presiona la combinación *control + y.*	2) El sistema busca en el historial el ultimo comando al que se le realizo la operación deshacer y en consecuencia ejecuta su opción rehacer. 3) El sistema muestra íntegramente el entorno de modelación en el estado posterior.
Flujos Alternos	
Acción del autor	**Respuesta del sistema**
Puntos de Extensión	
Poscondiciones:	El entorno de modelación vuelve a su estado posterior.

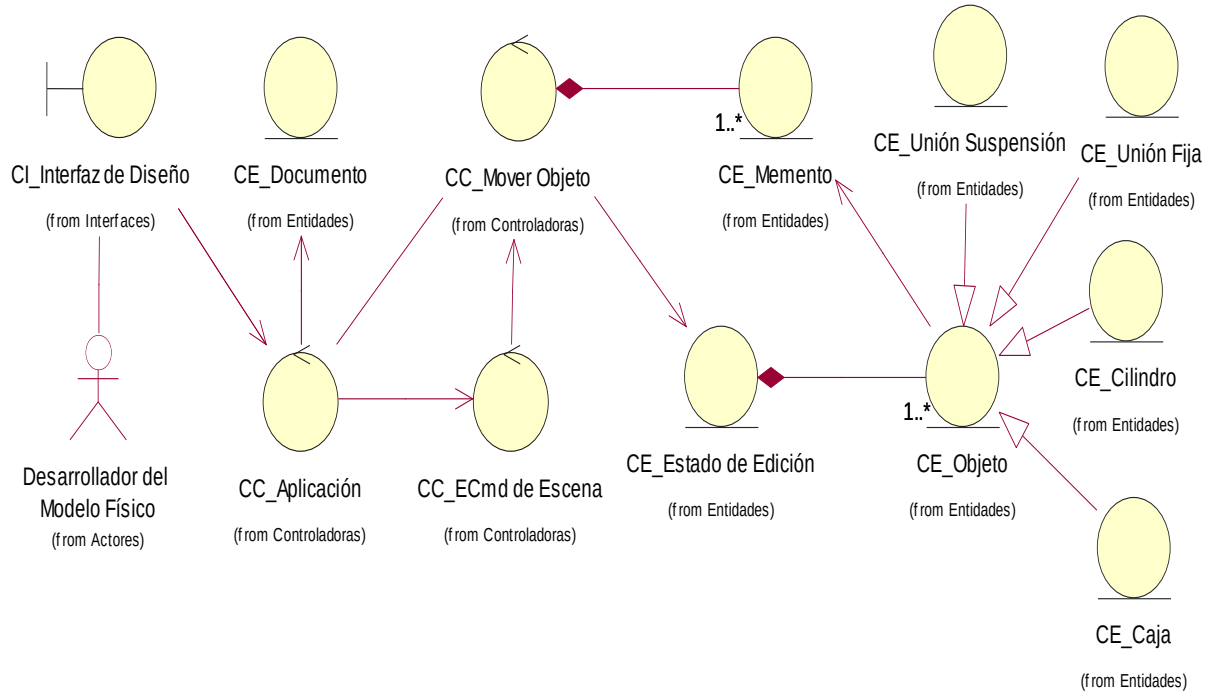
CU-16	**Detener Simulación**
Propósito	Detener la simulación en curso.
Actores	Diseñador del Modelo Físico.
Resumen: El Diseñador del Modelo Físico detiene la simulación en curso seleccionando realizar dicha acción en la barra de herramientas de la aplicación. El sistema articulado regresa a su posición inicial.	
Referencias:	RF: 21
Precondiciones	Existe una simulación en curso. El diseñador seleccionó la opción detener simulación.
Acción del autor	**Respuesta del sistema**
1) El Diseñador del Modelo Físico selecciona la opción detener la simulación en curso.	2) Detiene la simulación. 3) El sistema chequea la existencia del archivo salvado con la posición inicial del sistema articulado. 4) En caso negativo muestra un mensaje de información incluyendo el nuevo estado: PAUSA.
Flujo Alterno 1	
Acción del autor	**Respuesta del sistema**
	3) En caso positivo se elimina el sistema articulado actual. 4) Se carga el sistema articulado desde el fichero esperado.
Puntos de Extensión: Flujo Alterno 1 Línea 4: Ver CU-12 Cargar Sistema Articulado.	
Poscondiciones	El sistema articulado regresa a su posición inicial.

Anexo 5. Modelo de clases del análisis

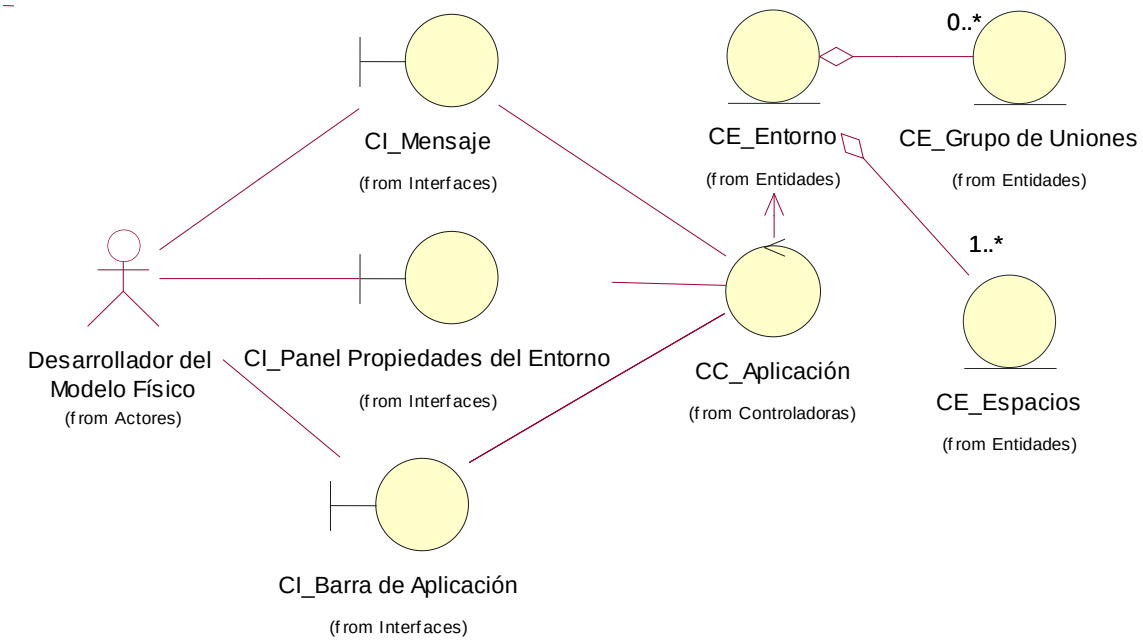
CU Cargar Sistema Articulado



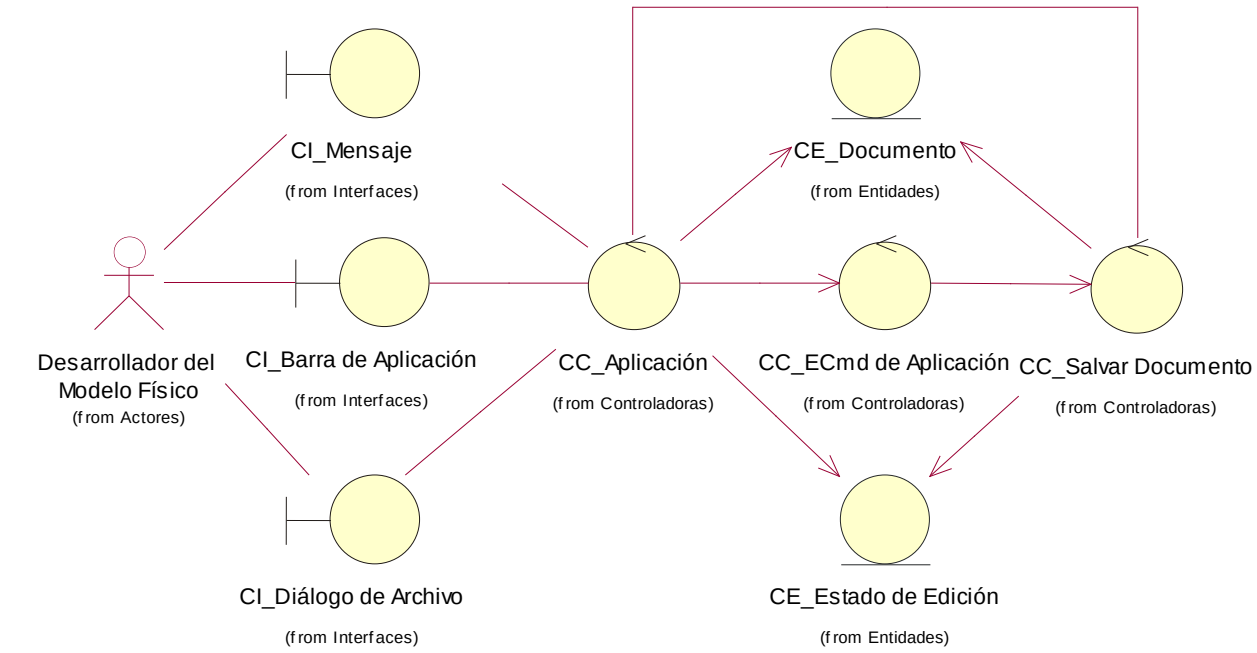
CU Configurar Orientación y Posición



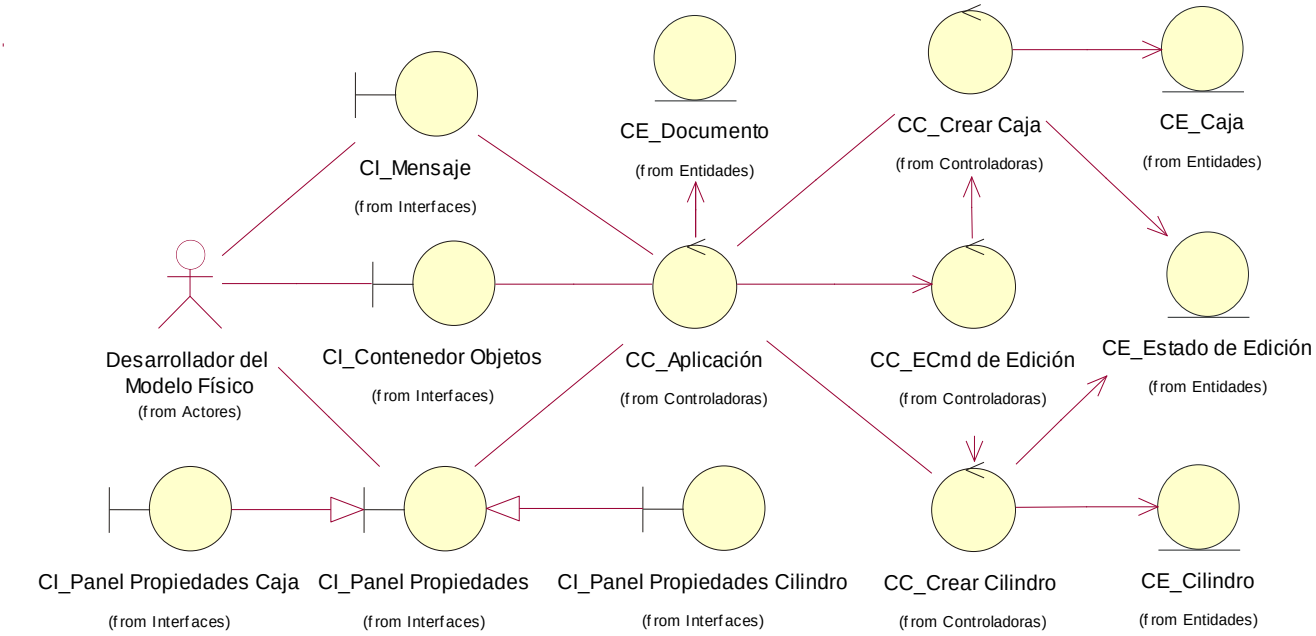
CU Configurar Propiedades del Entorno



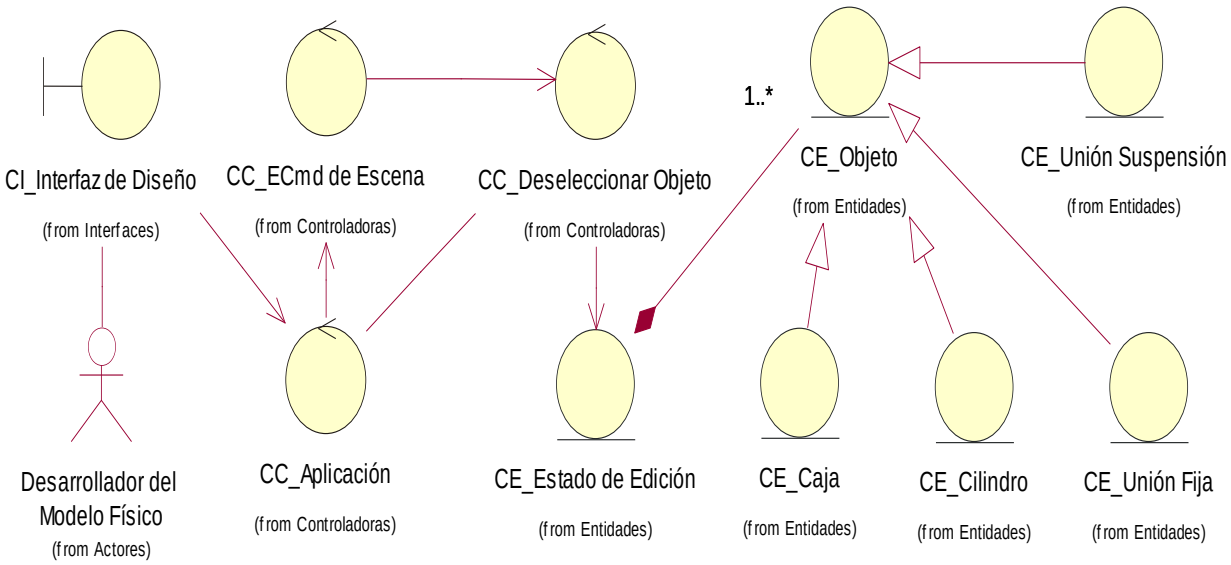
CU Crear Nuevo Documento



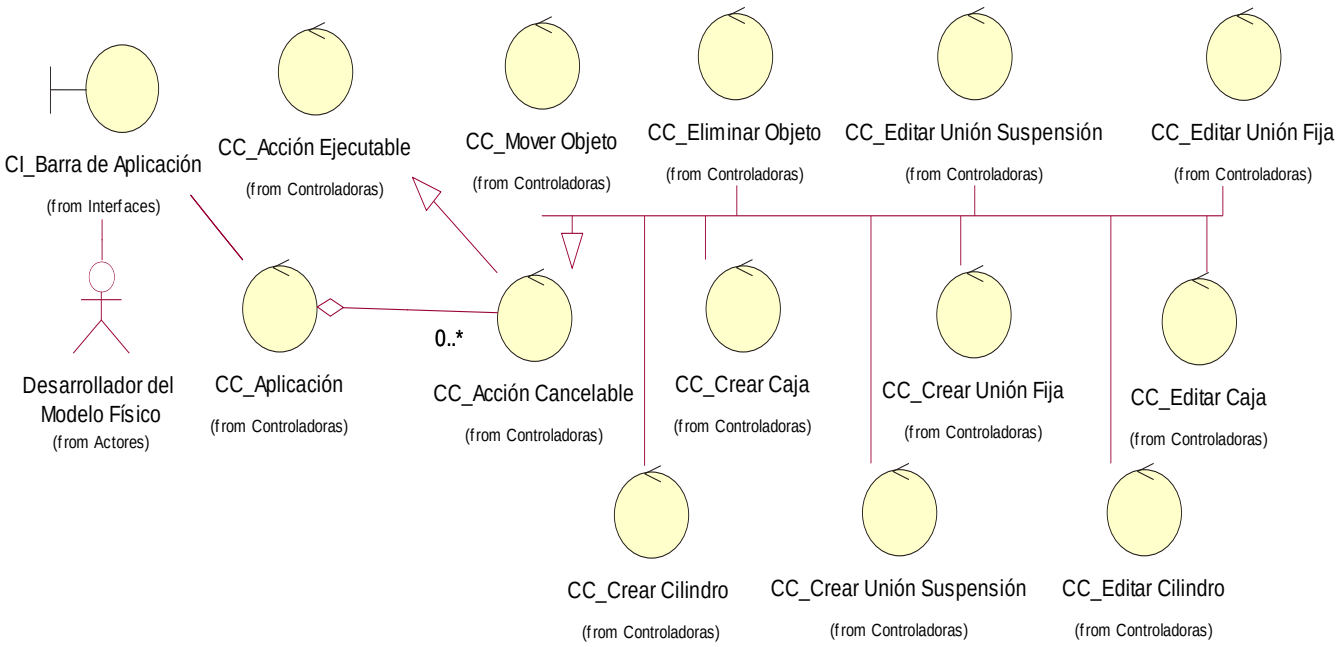
CU Crear Primitiva Geométrica



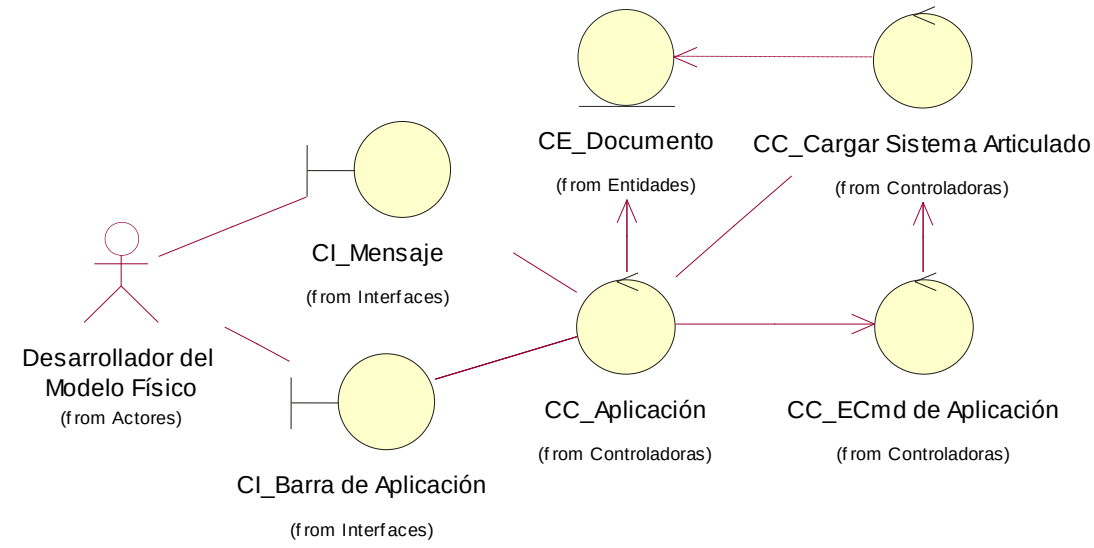
CU Deseleccionar Objeto



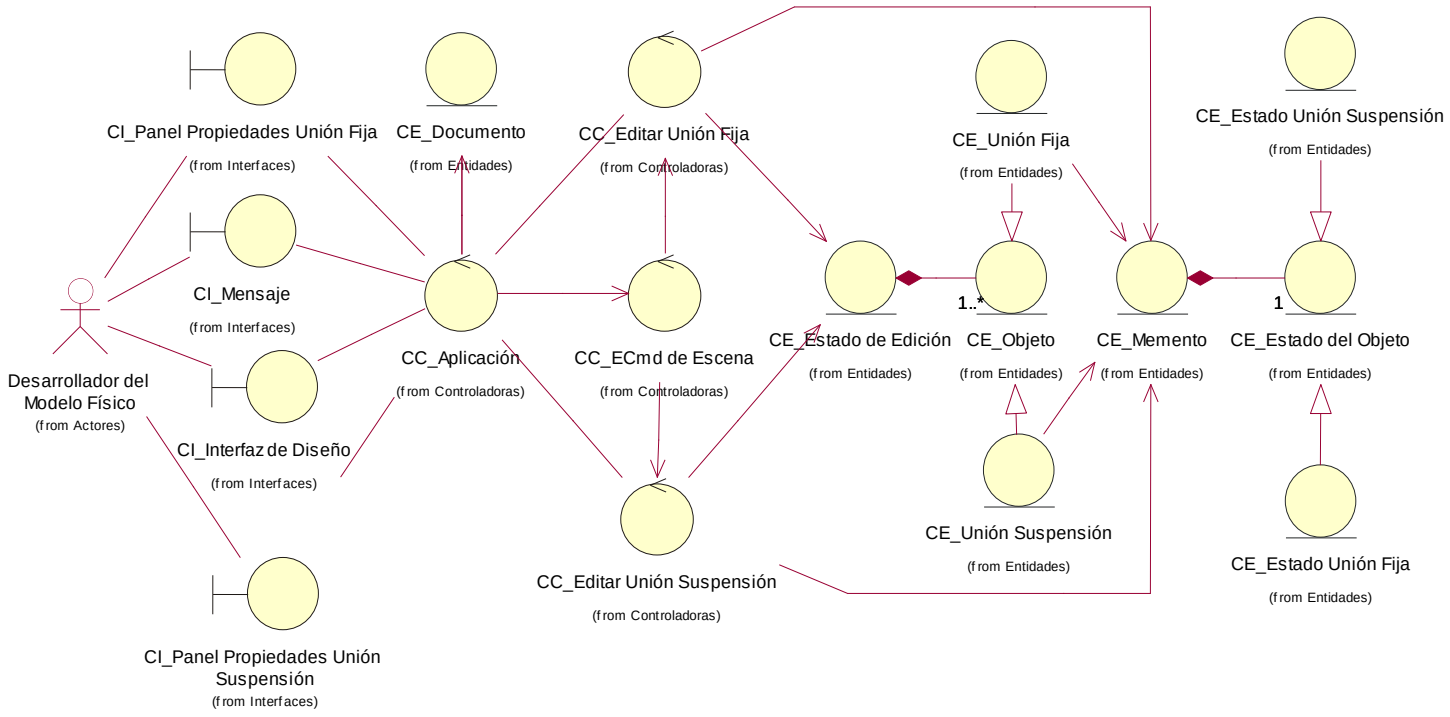
CU Deshacer un cambio



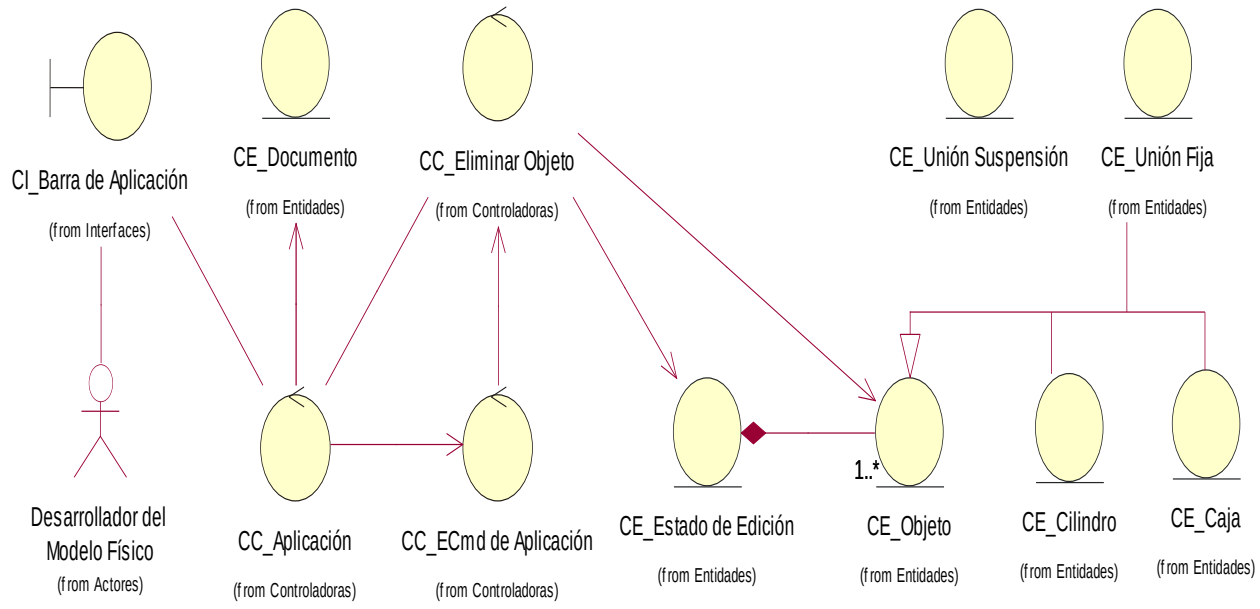
CU Detener Simulación



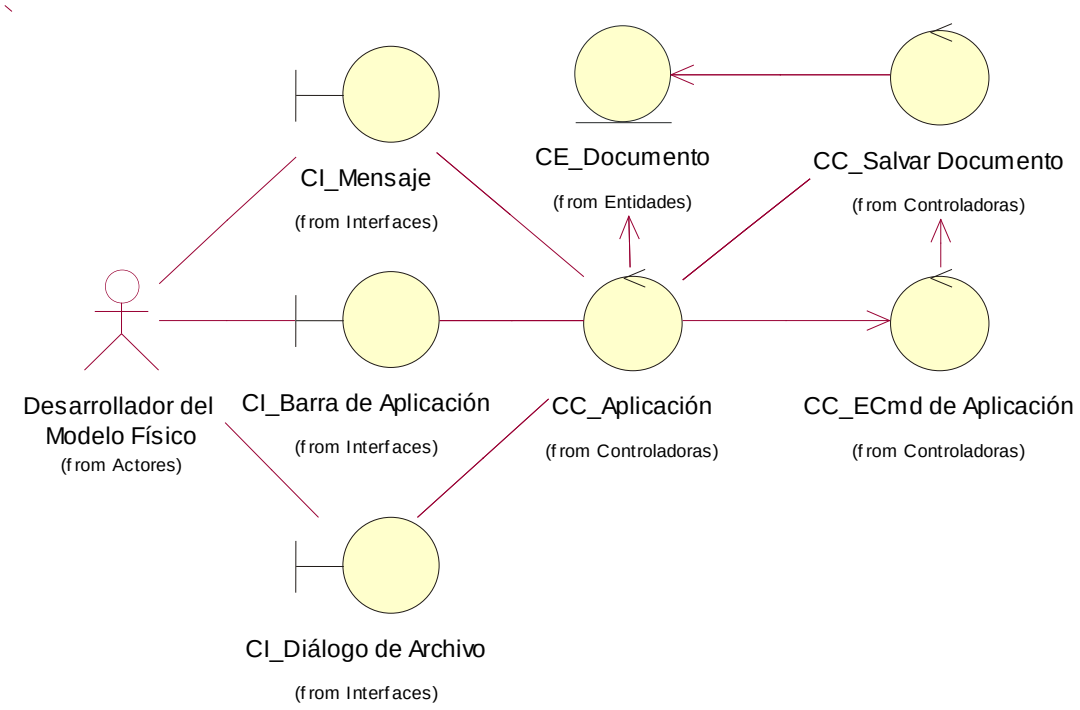
CU Editar Unión



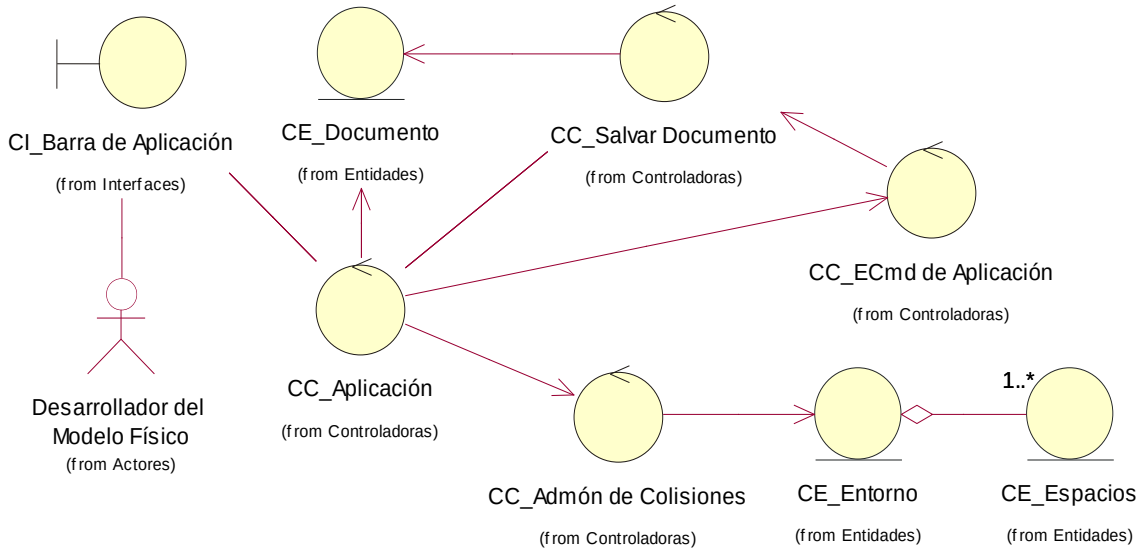
CU Eliminar Objeto



CU Salvar Sistema Articulado

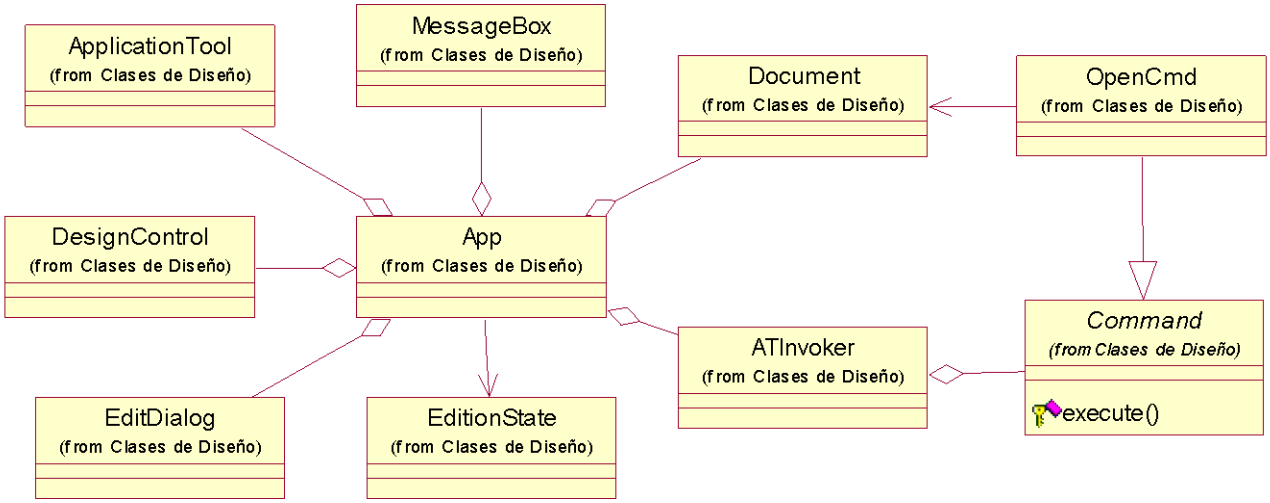


CU Simular Sistema Articulado

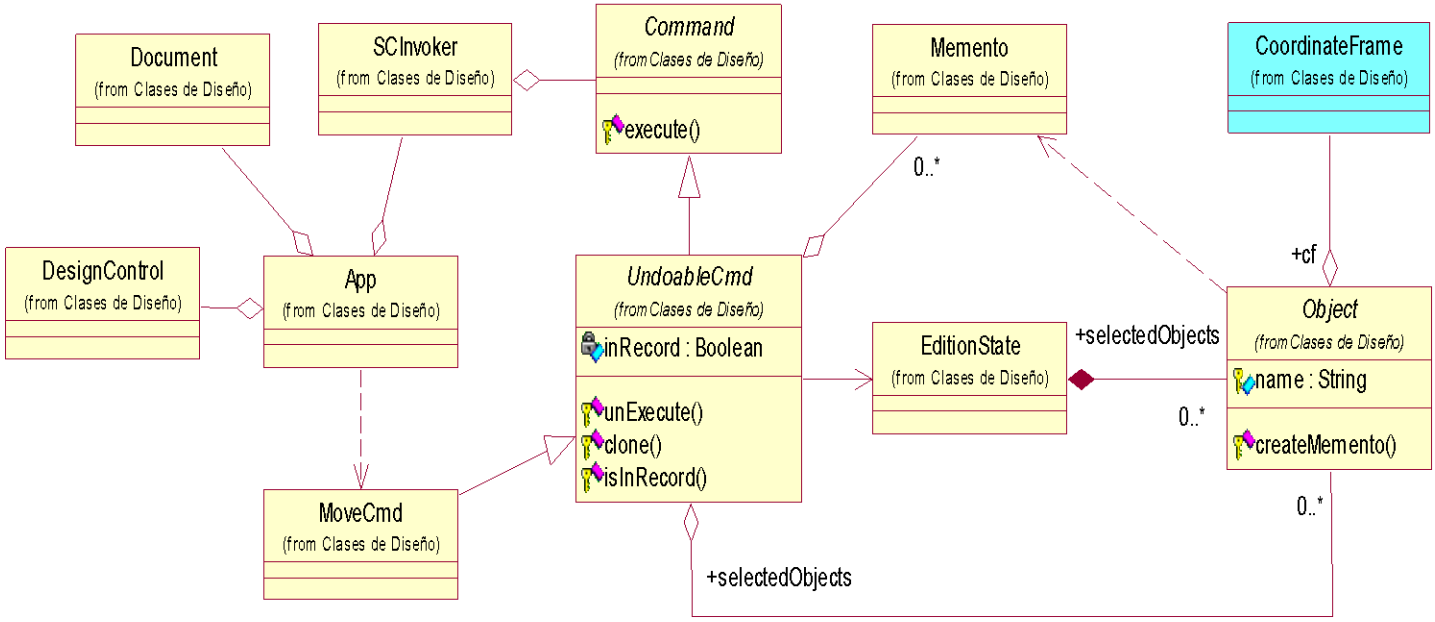


Anexo 6. Modelo de clases del diseño

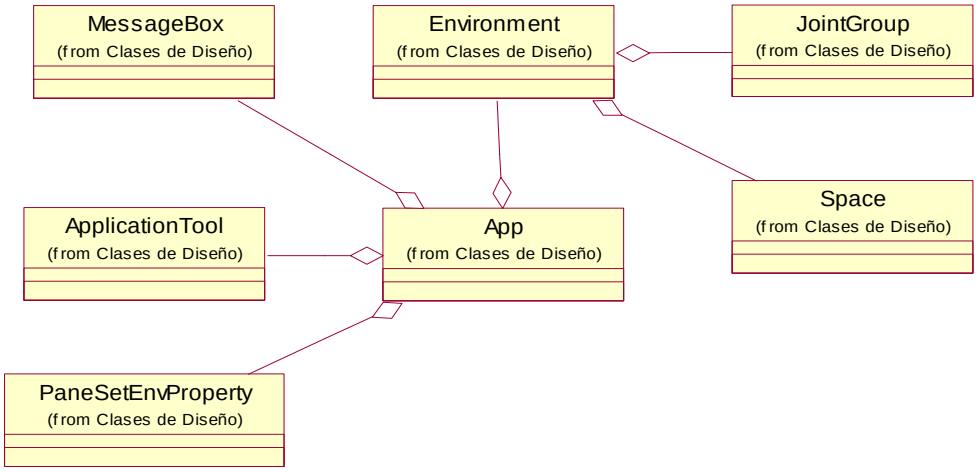
CU Cargar Sistema Articulado



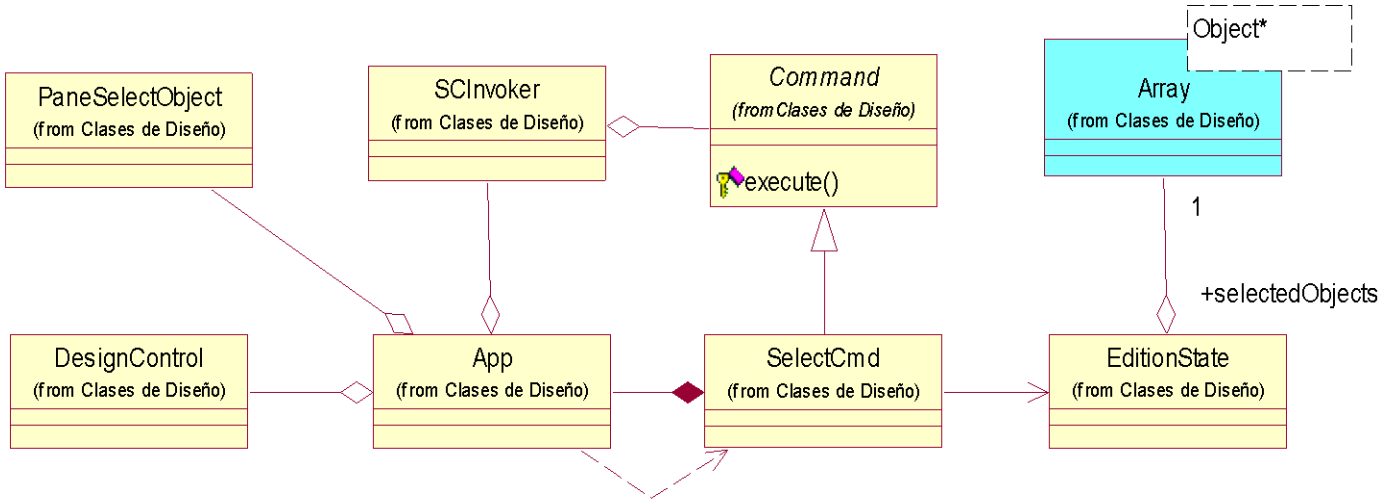
CU Configurar Orientación y Posición



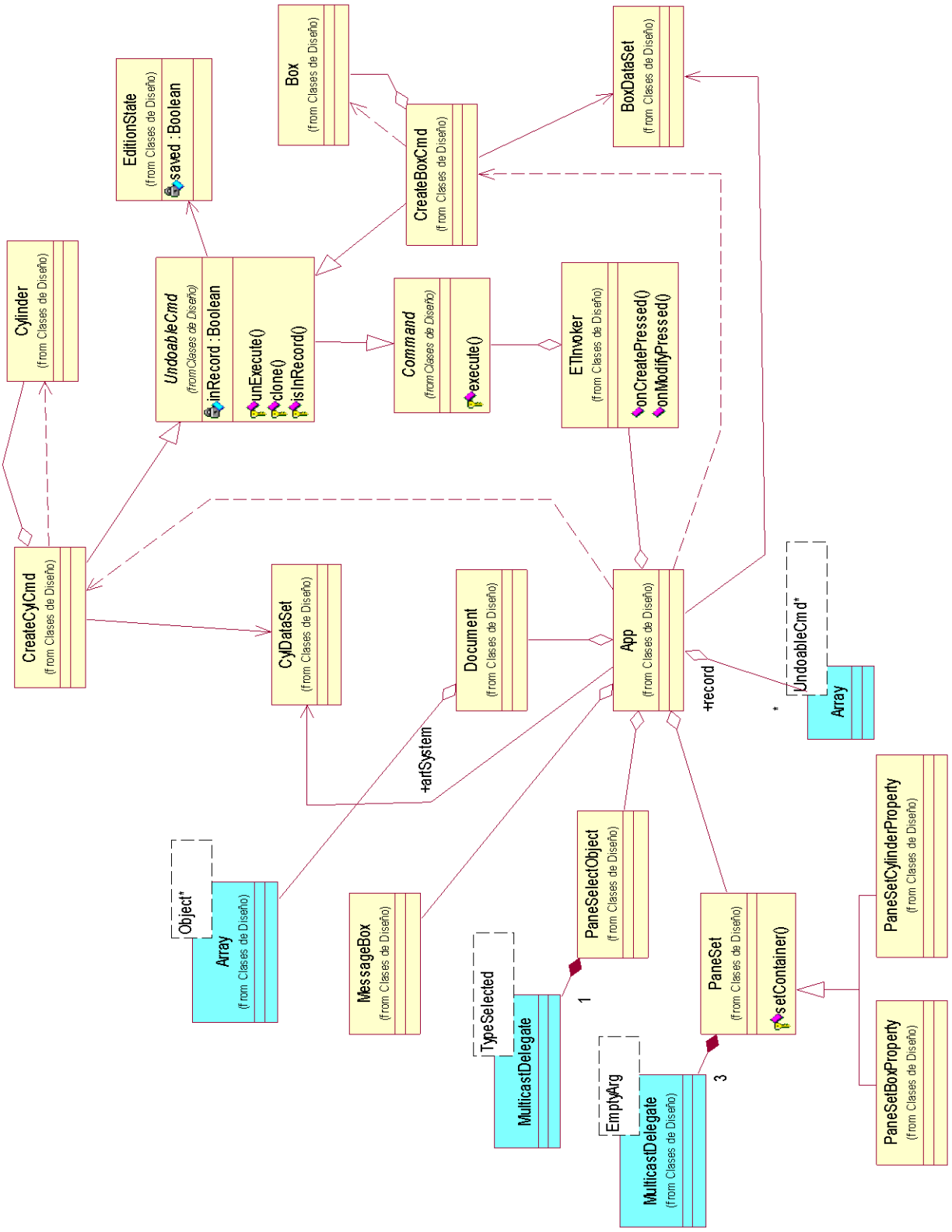
CU Configurar Propiedades del Entorno



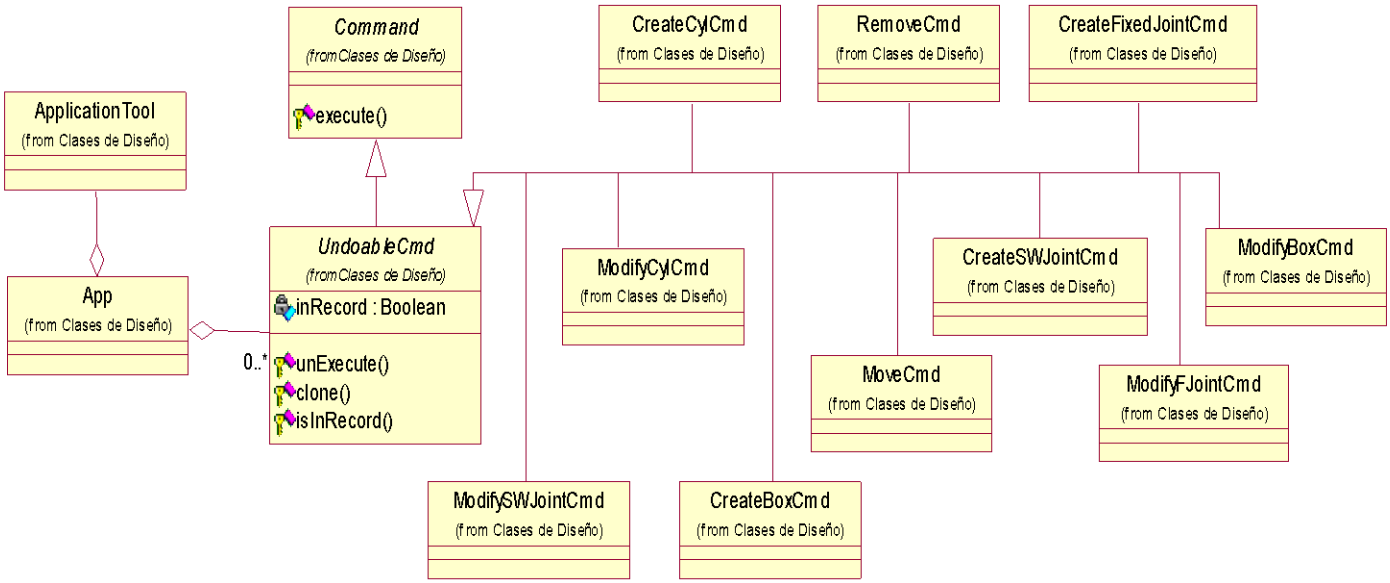
CU Seleccionar Objetos de la Escena



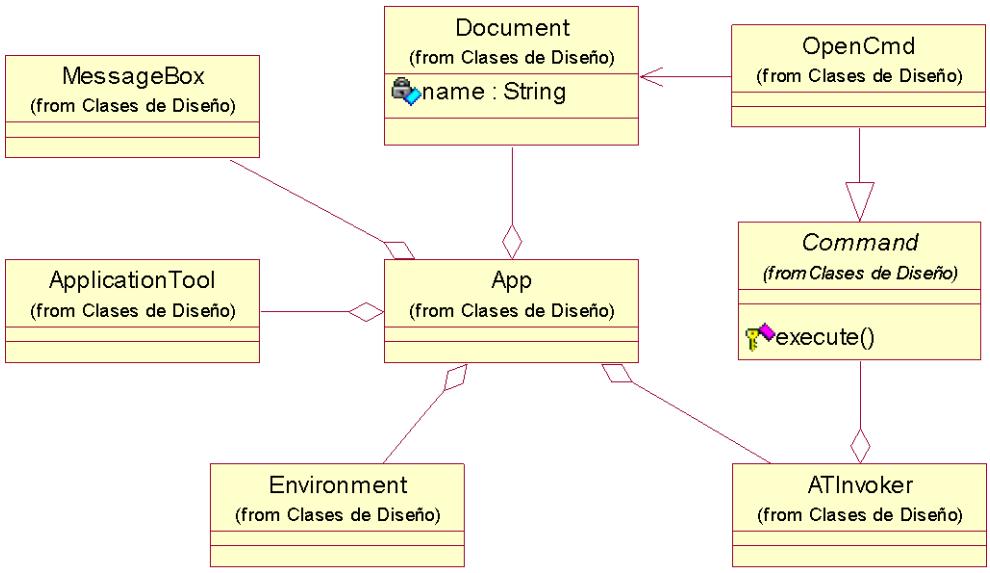
CU Crear Primitiva Geométrica



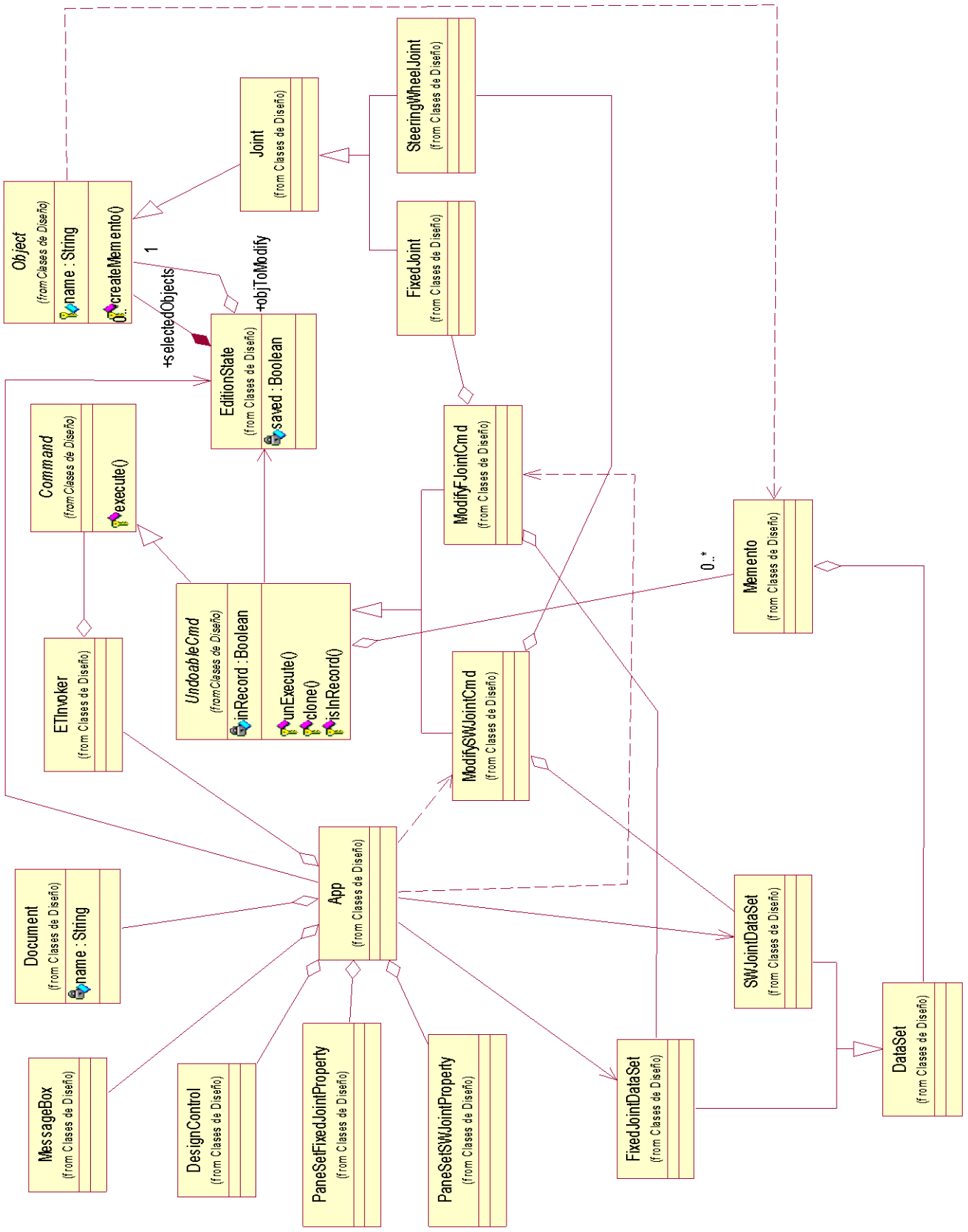
CU Rehacer un cambio



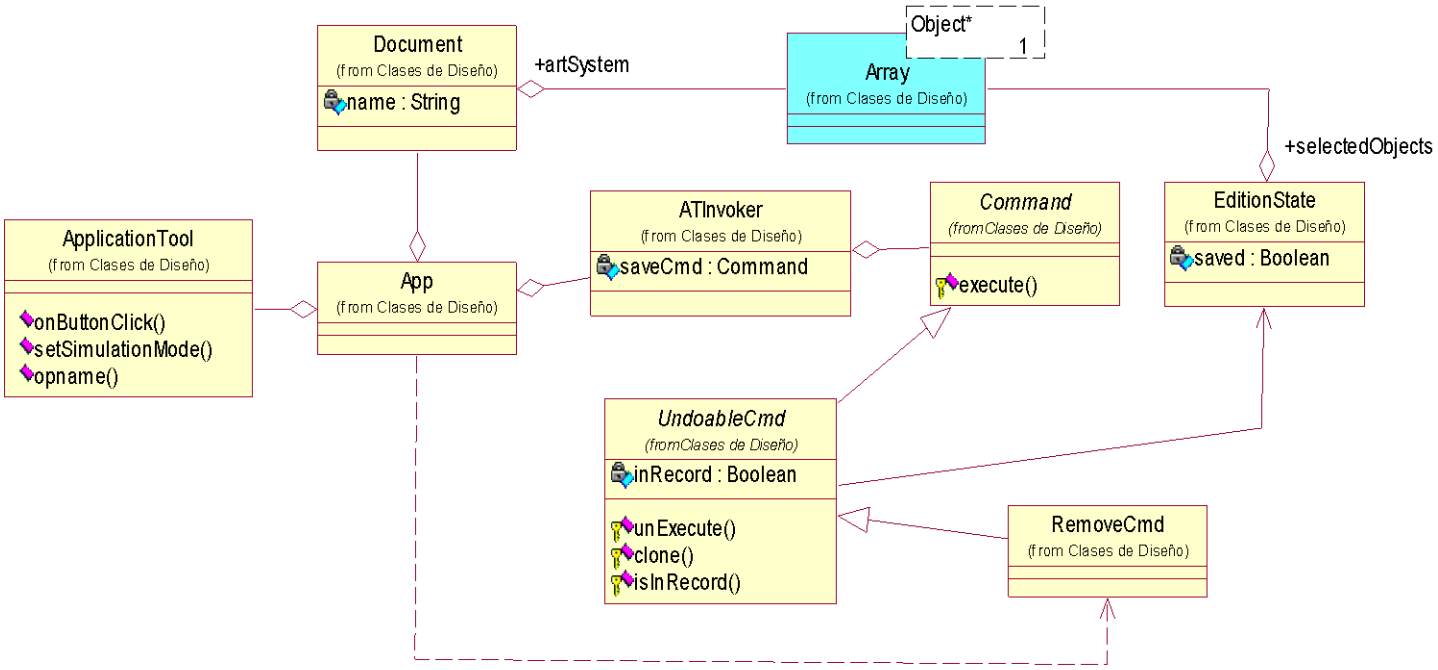
CU Detener la simulación



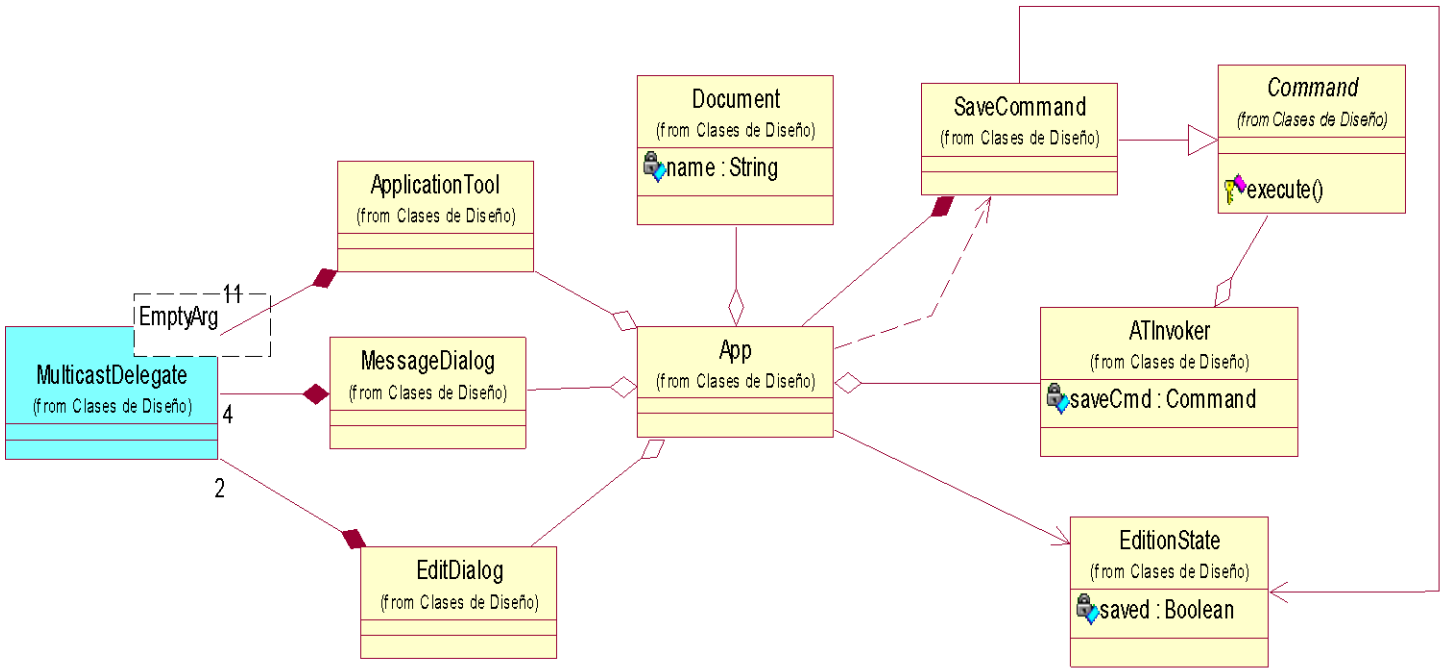
CU Editar Unión



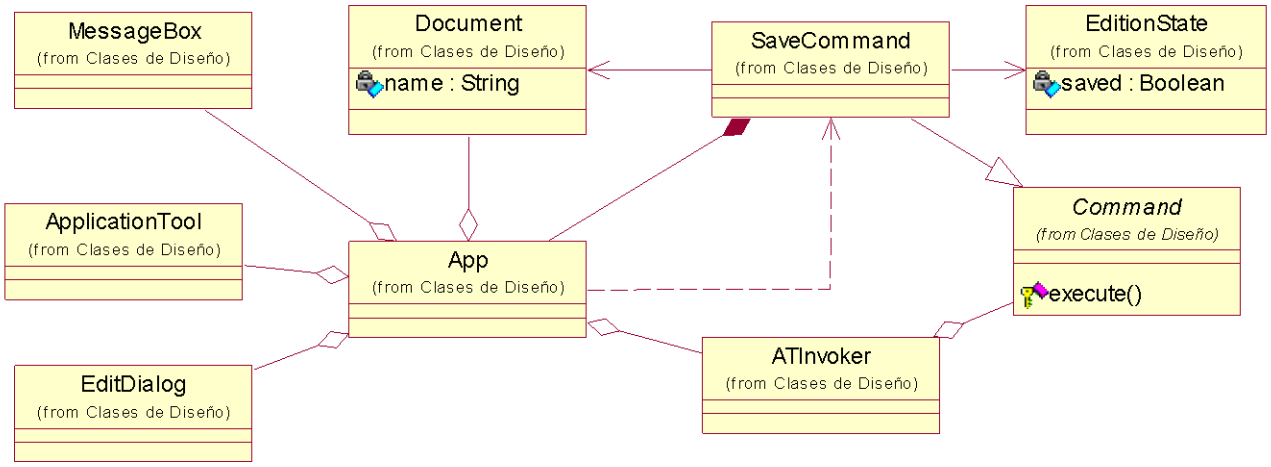
CU Eliminar Objeto de la Escena



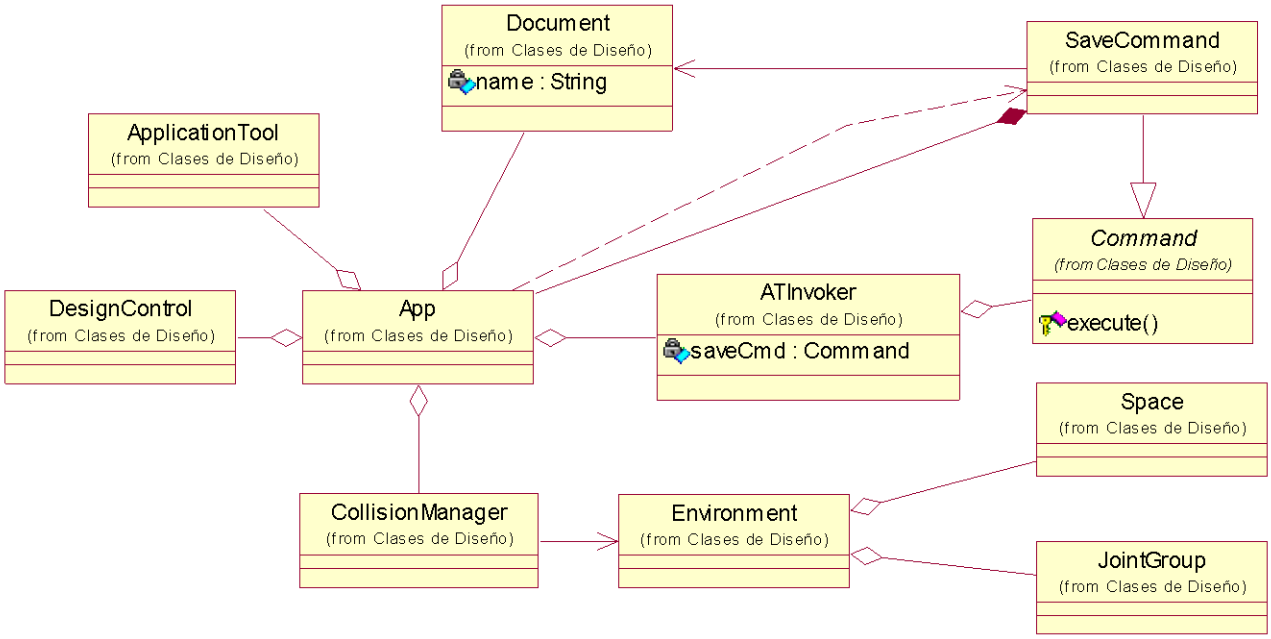
CU Nuevo Documento



CU Salvar Sistema Articulado

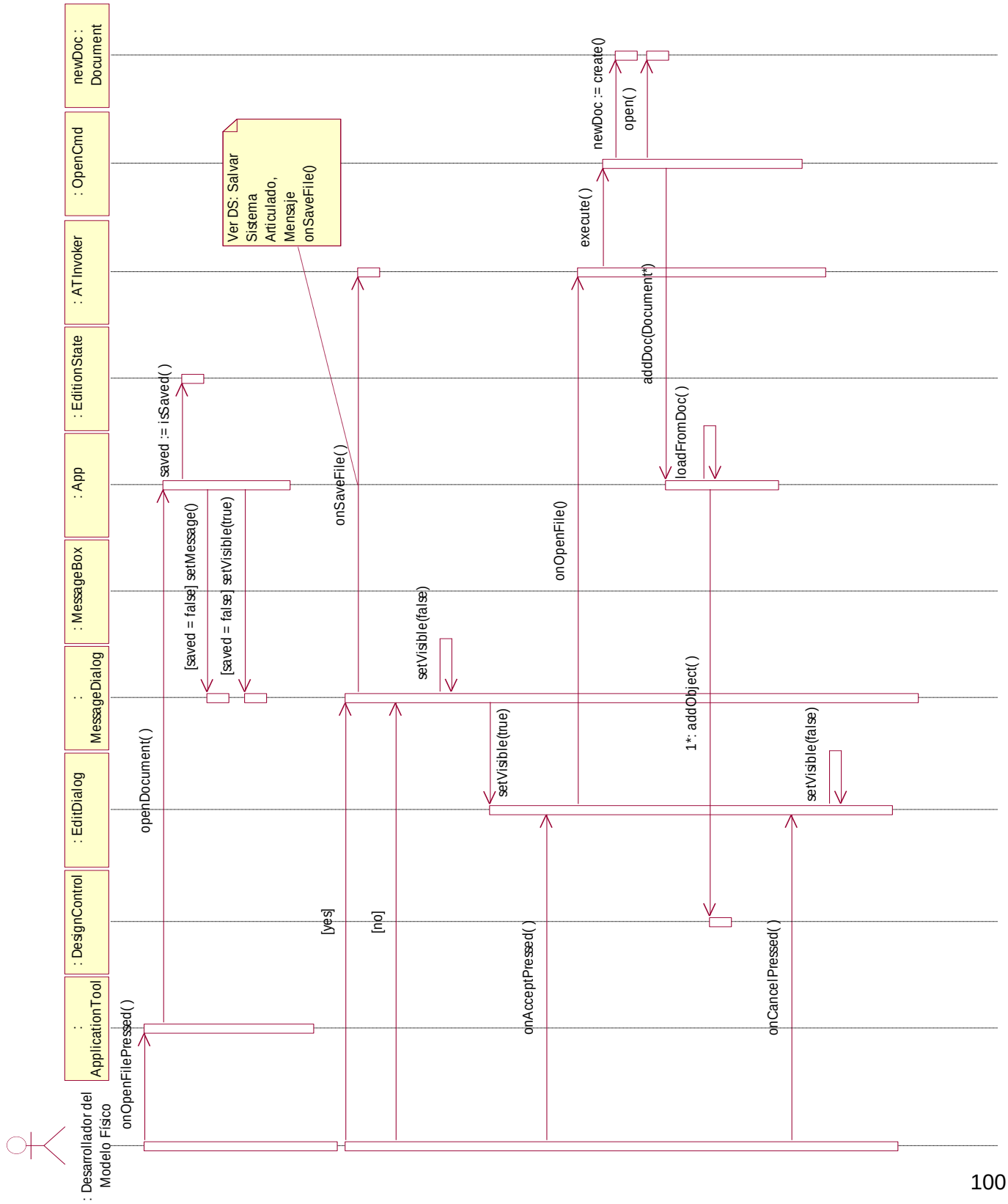


CU Simular Sistema Articulado

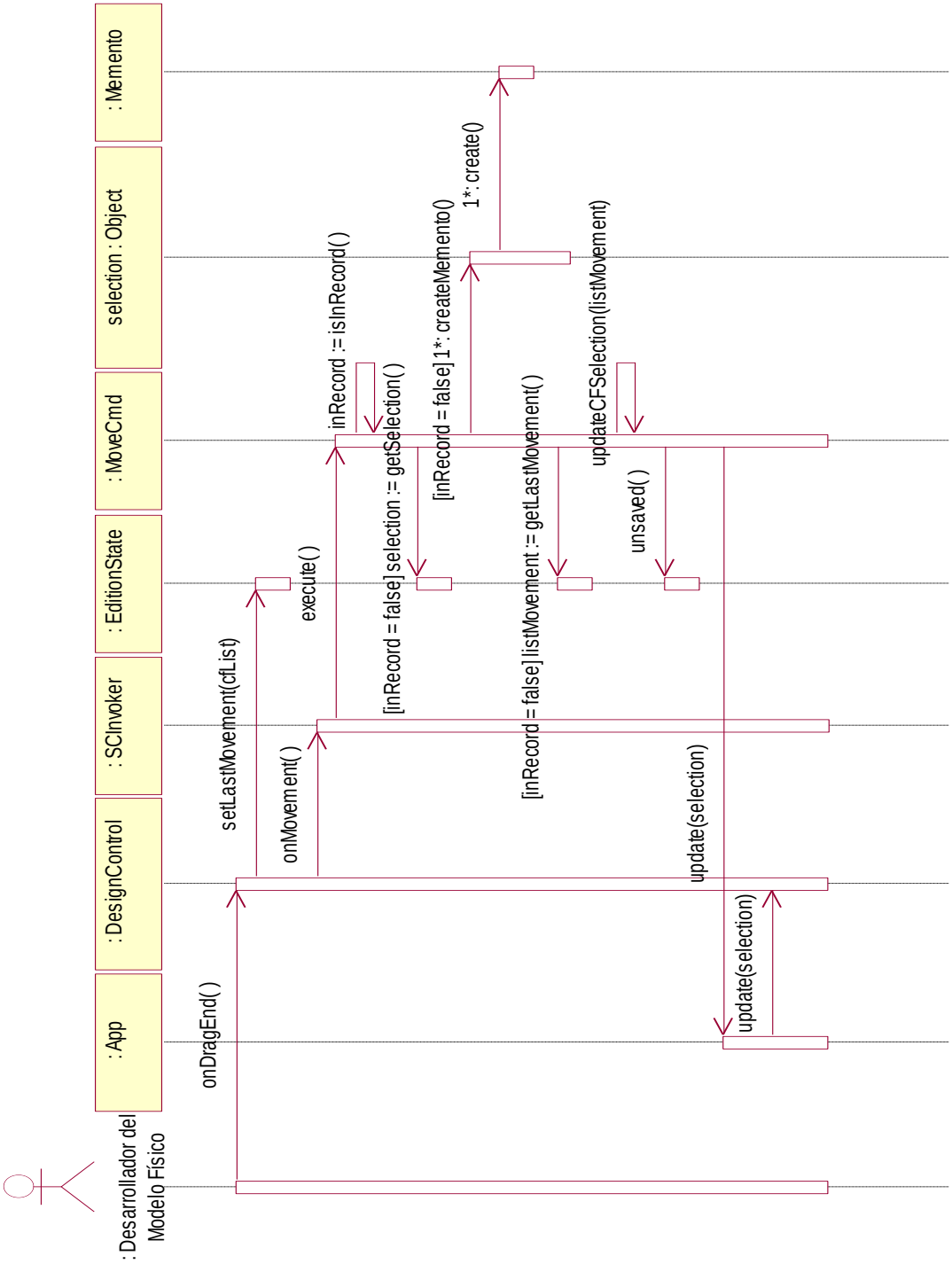


Anexo 7. Diagramas de Secuencia

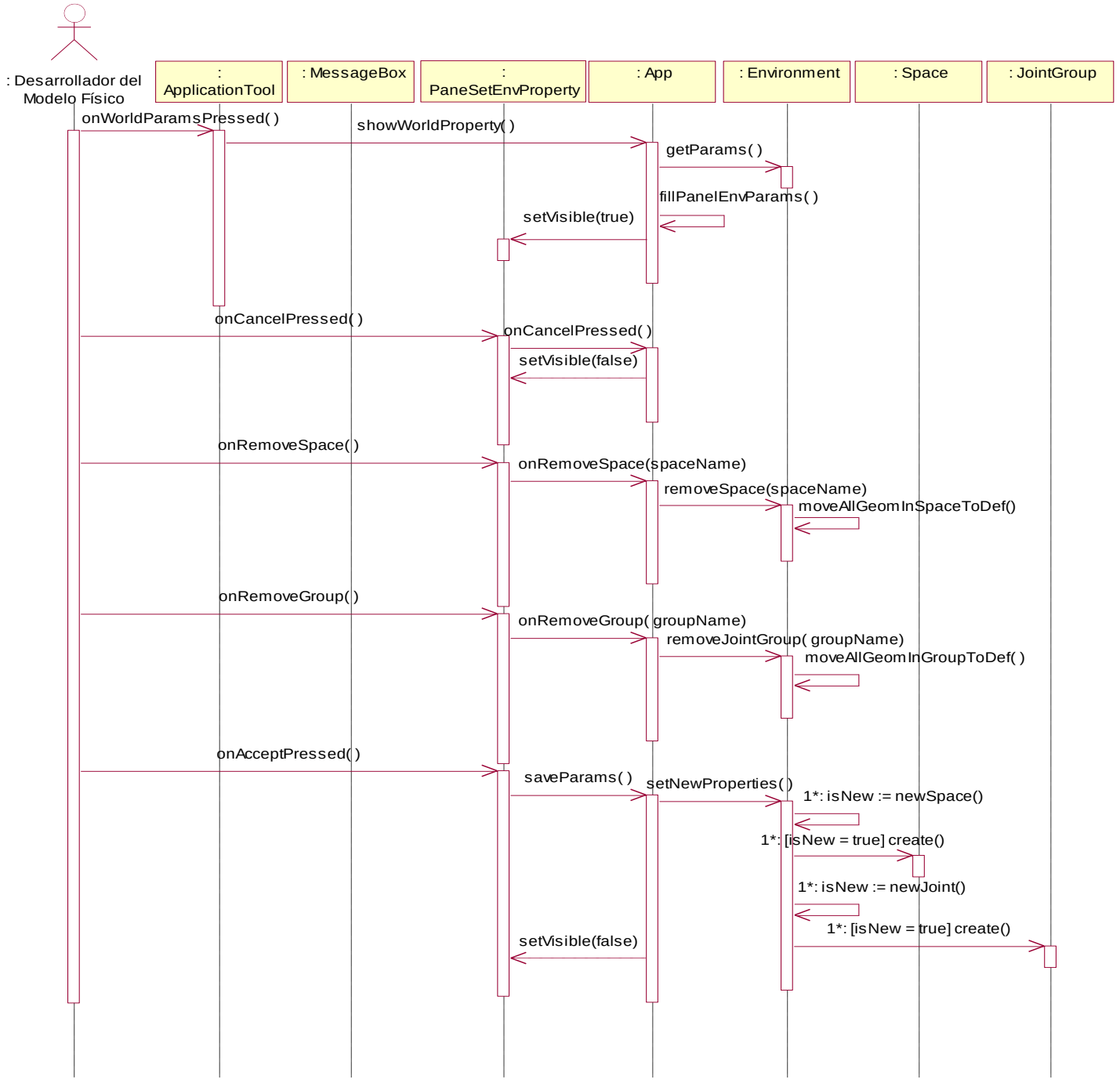
CU Cargar Sistema Articulado



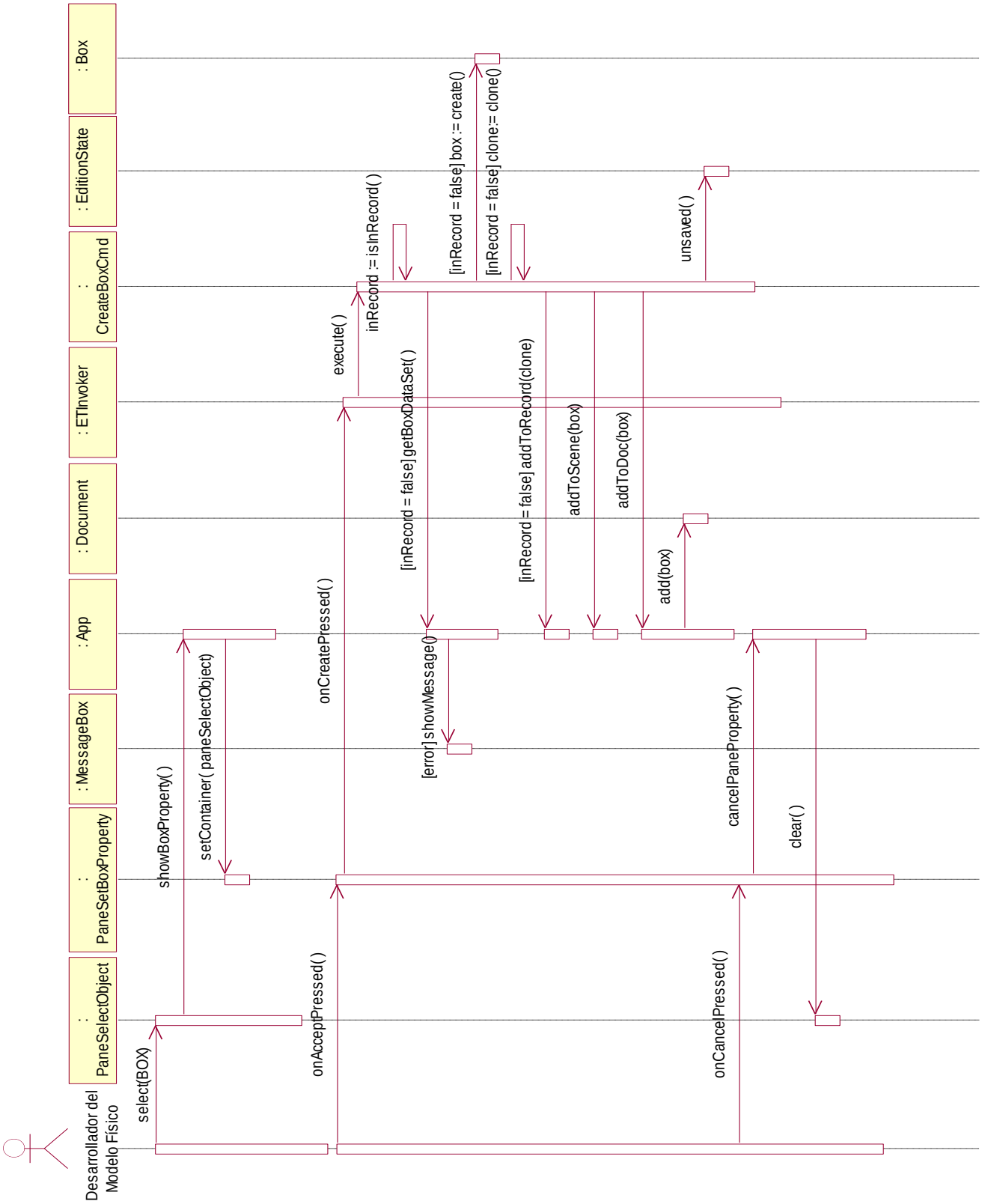
CU Configurar Orientación y Posición



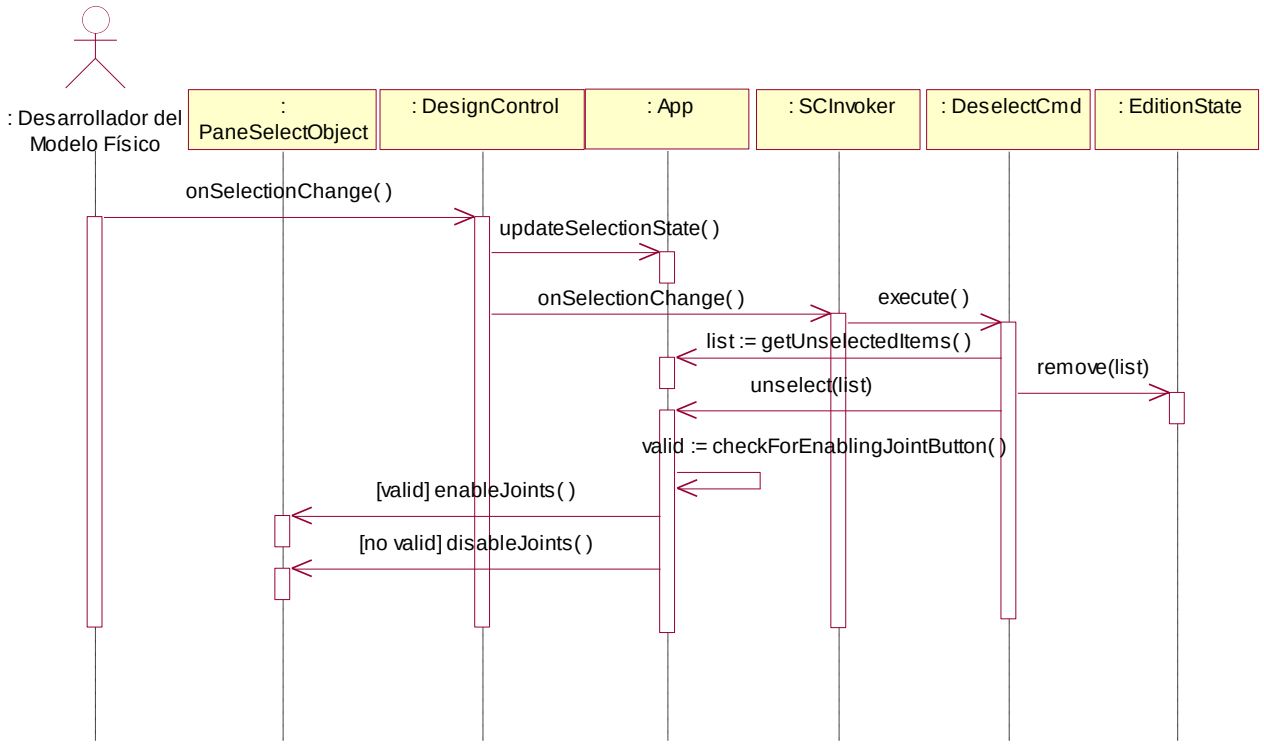
CU Configurar Propiedades del Entorno



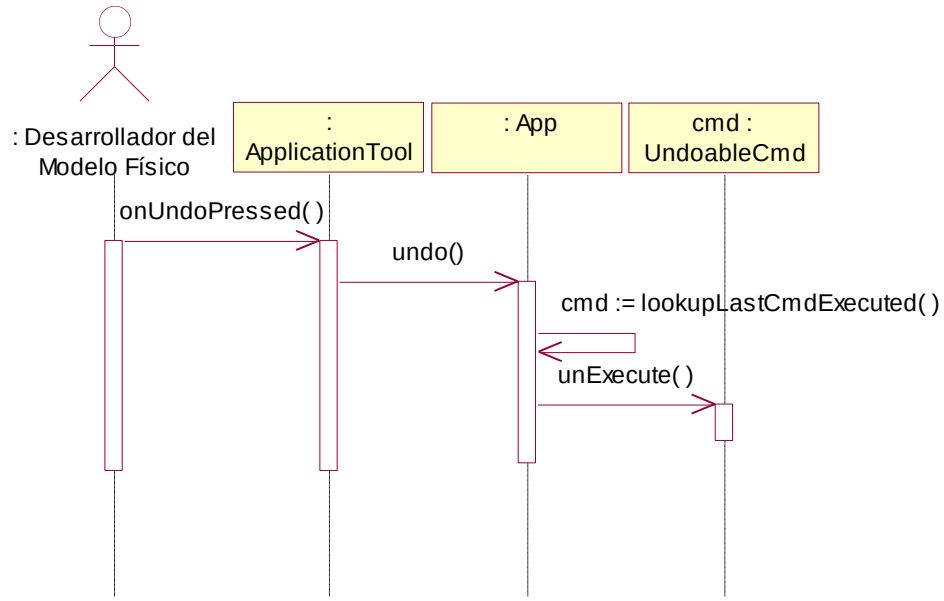
CU Crear Primitiva Geométrica. Escenario Crear Caja



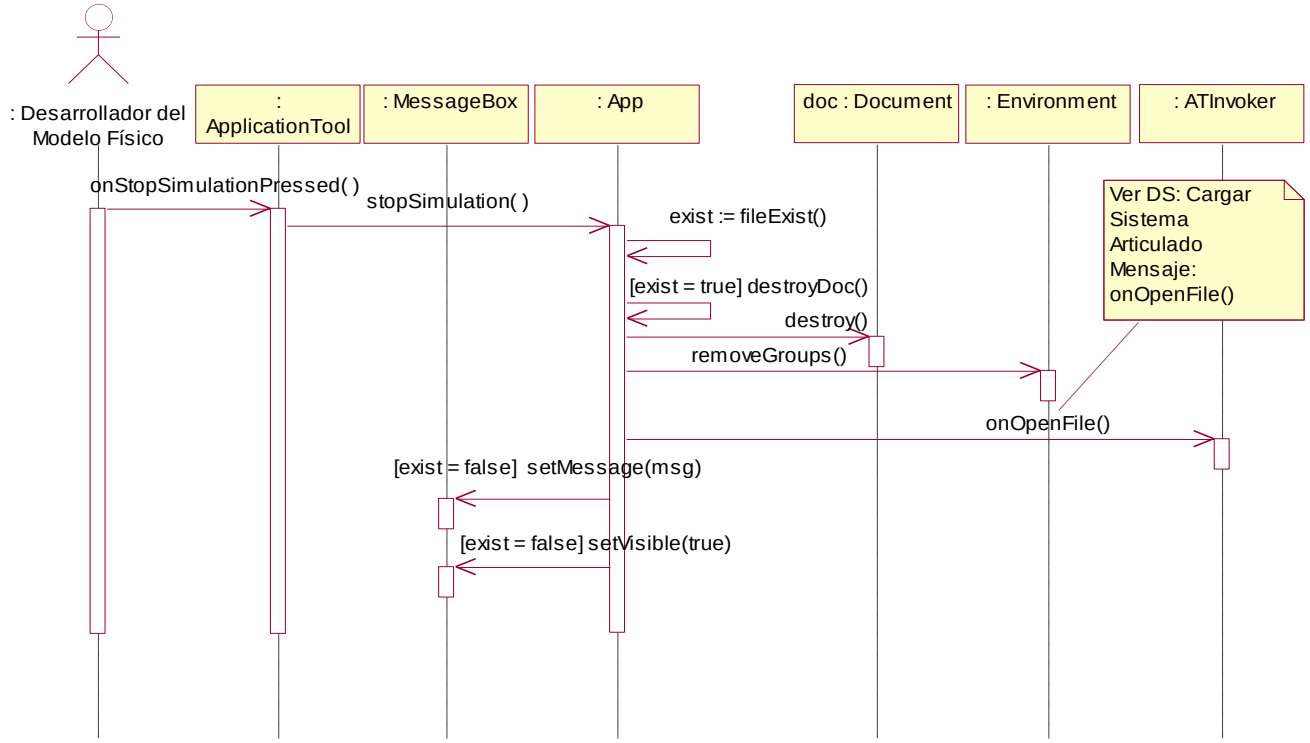
CU Deseleccionar Objetos de la Escena



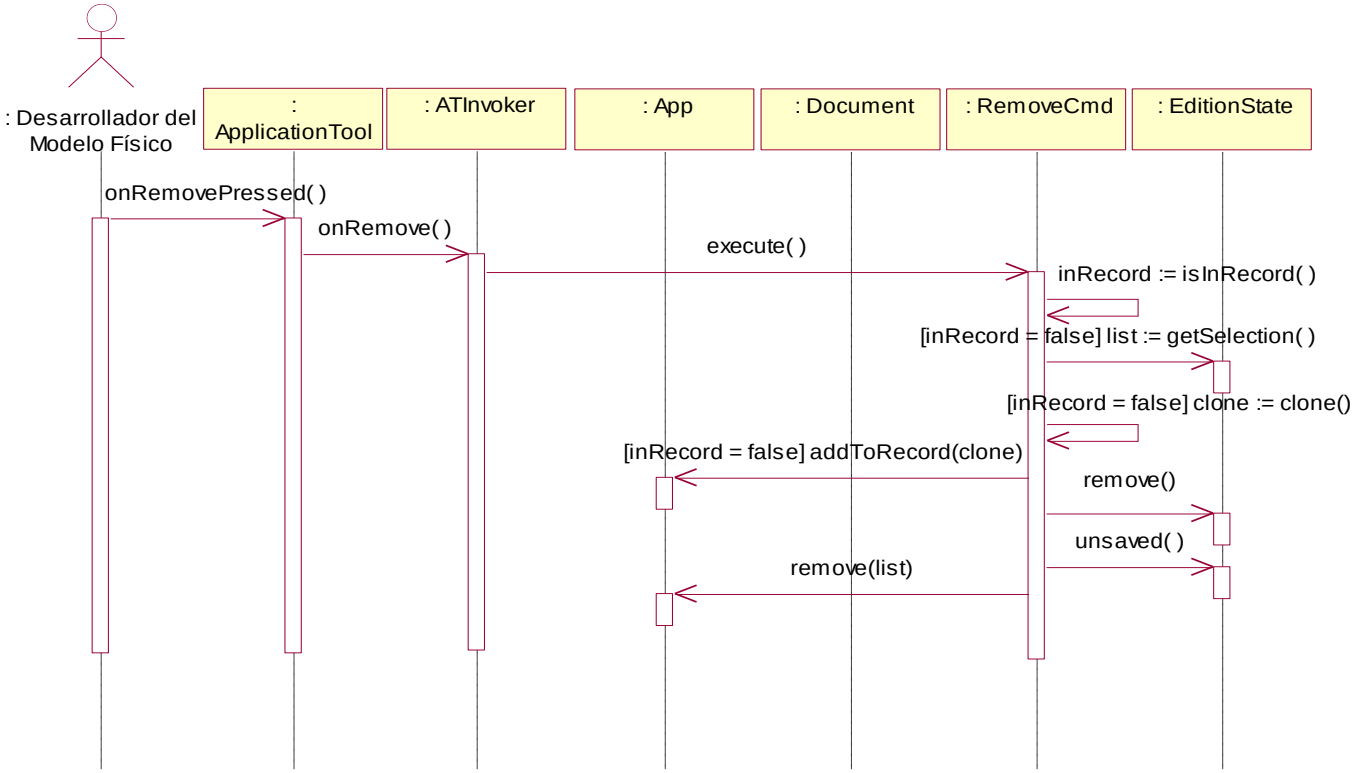
CU Deshacer Un Cambio



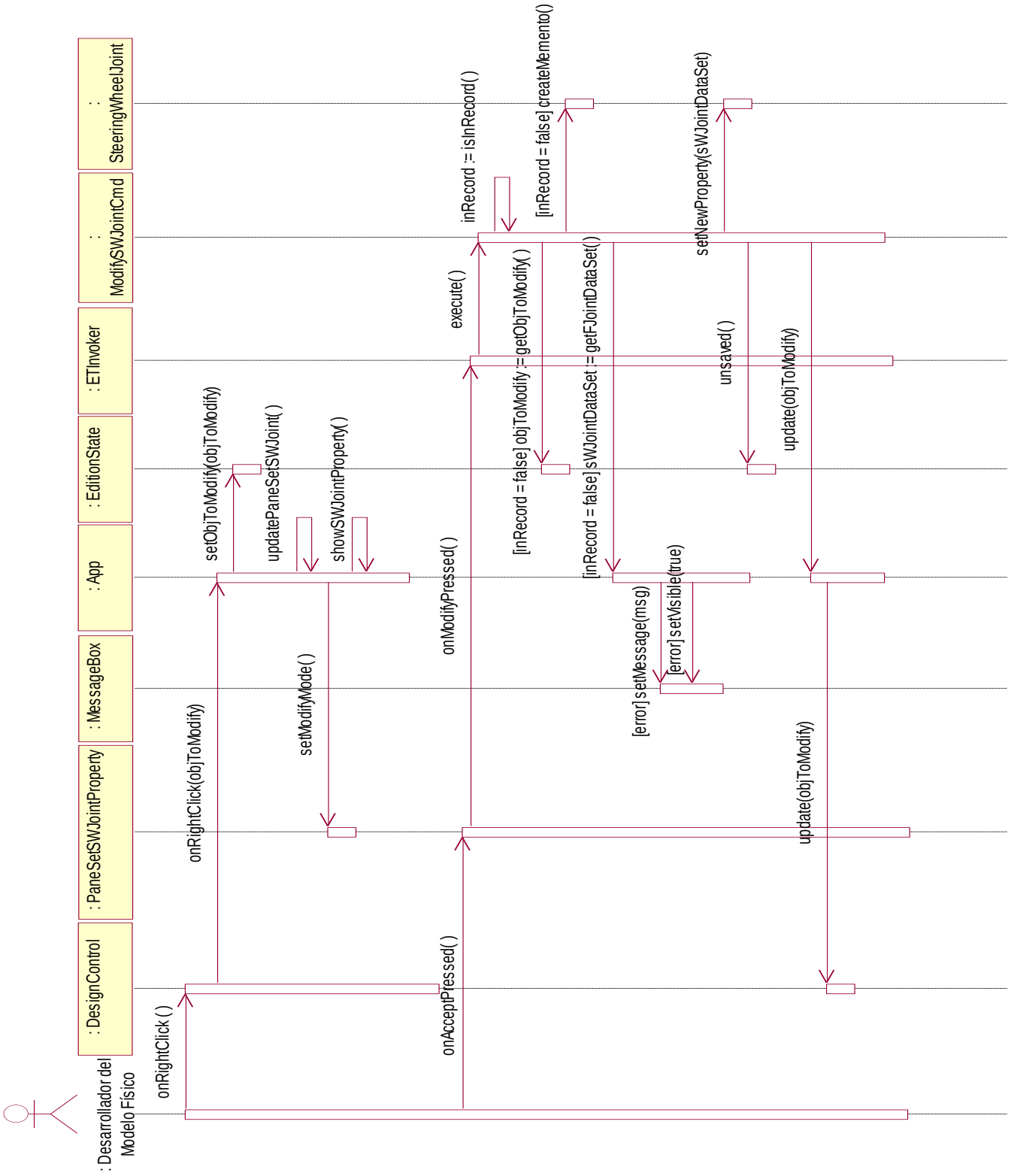
CU Detener la Simulación



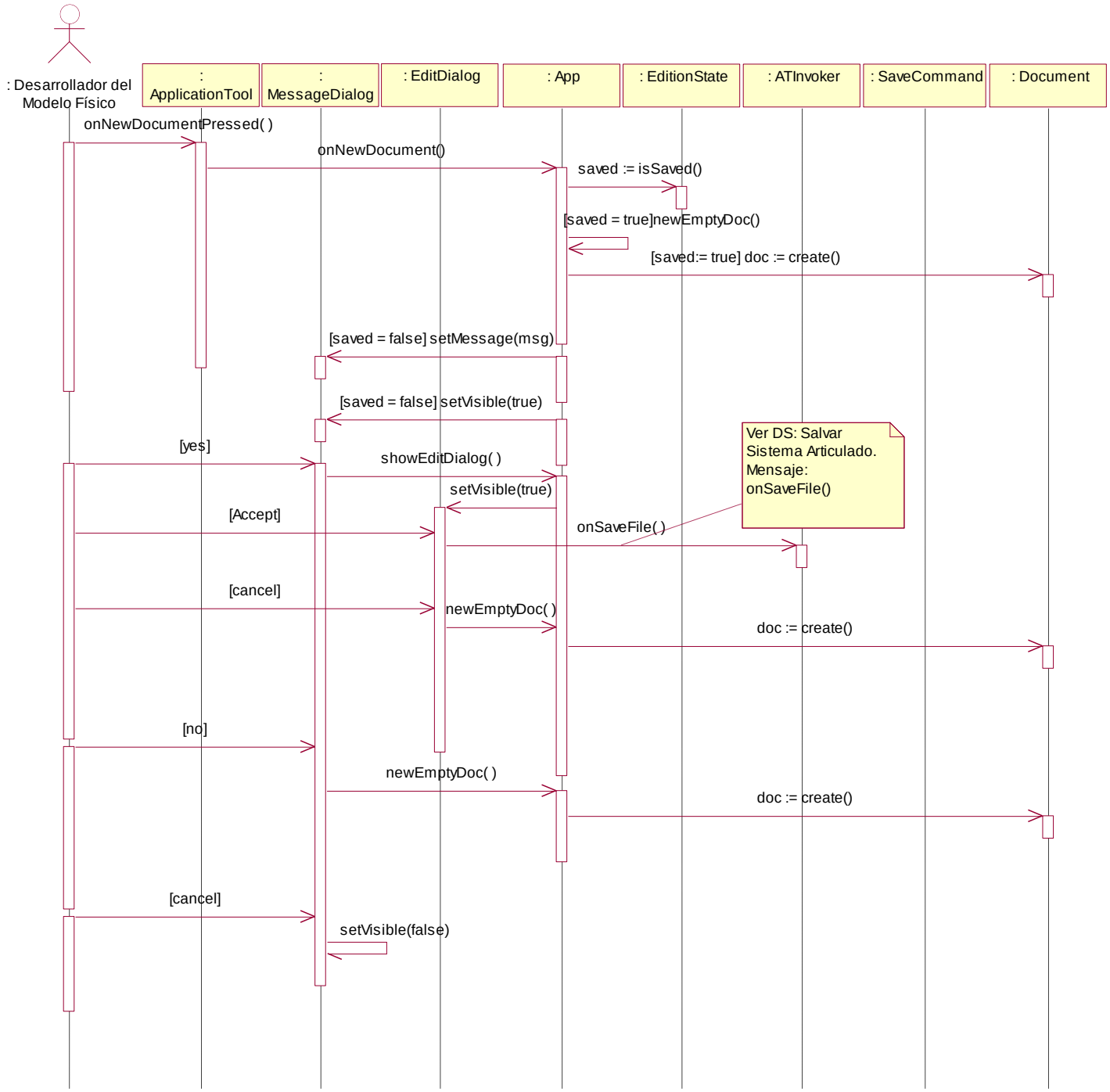
CU Eliminar Objeto de la Escena



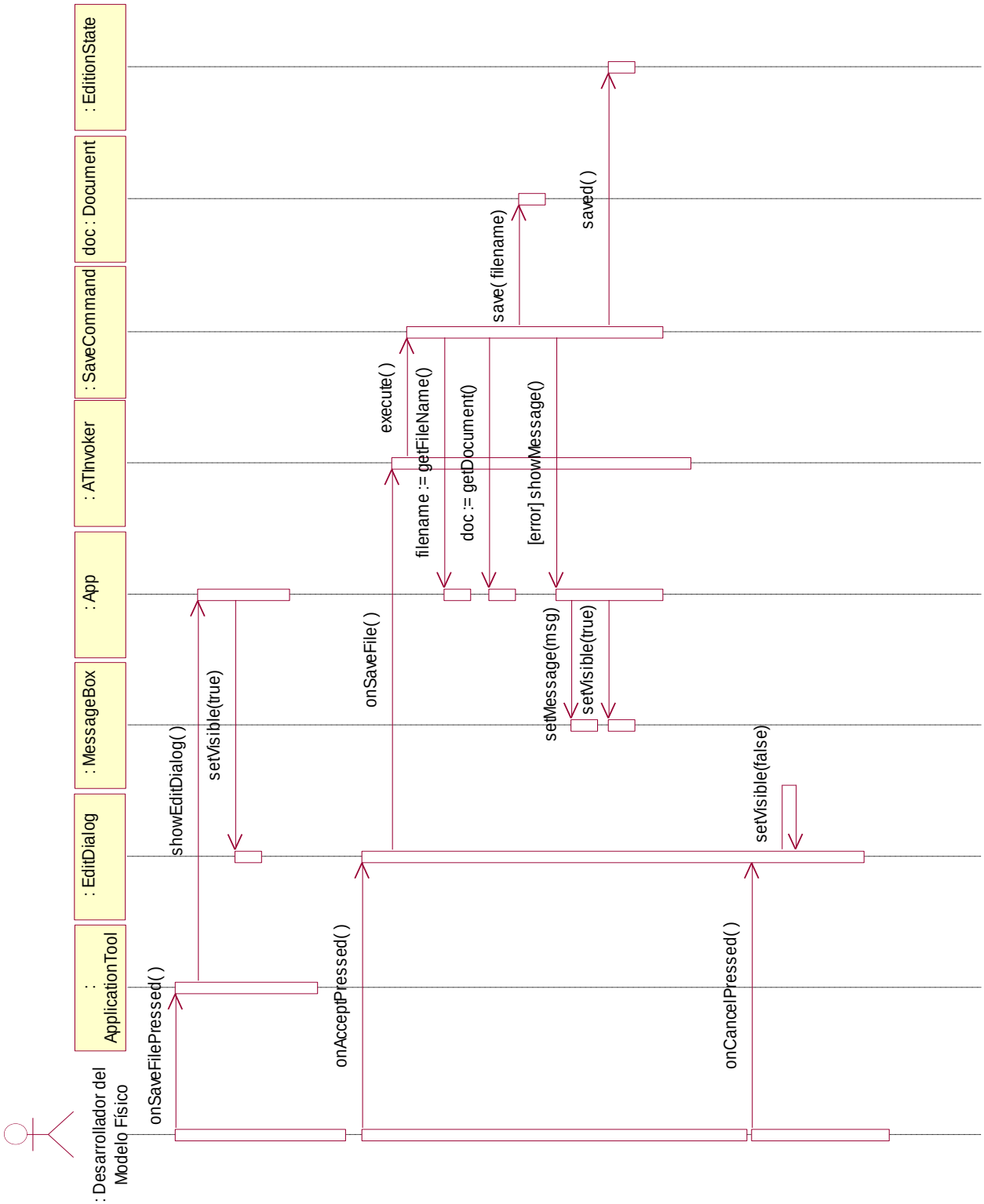
CU Editar Unión. Escenario Editar Unión con Suspensión



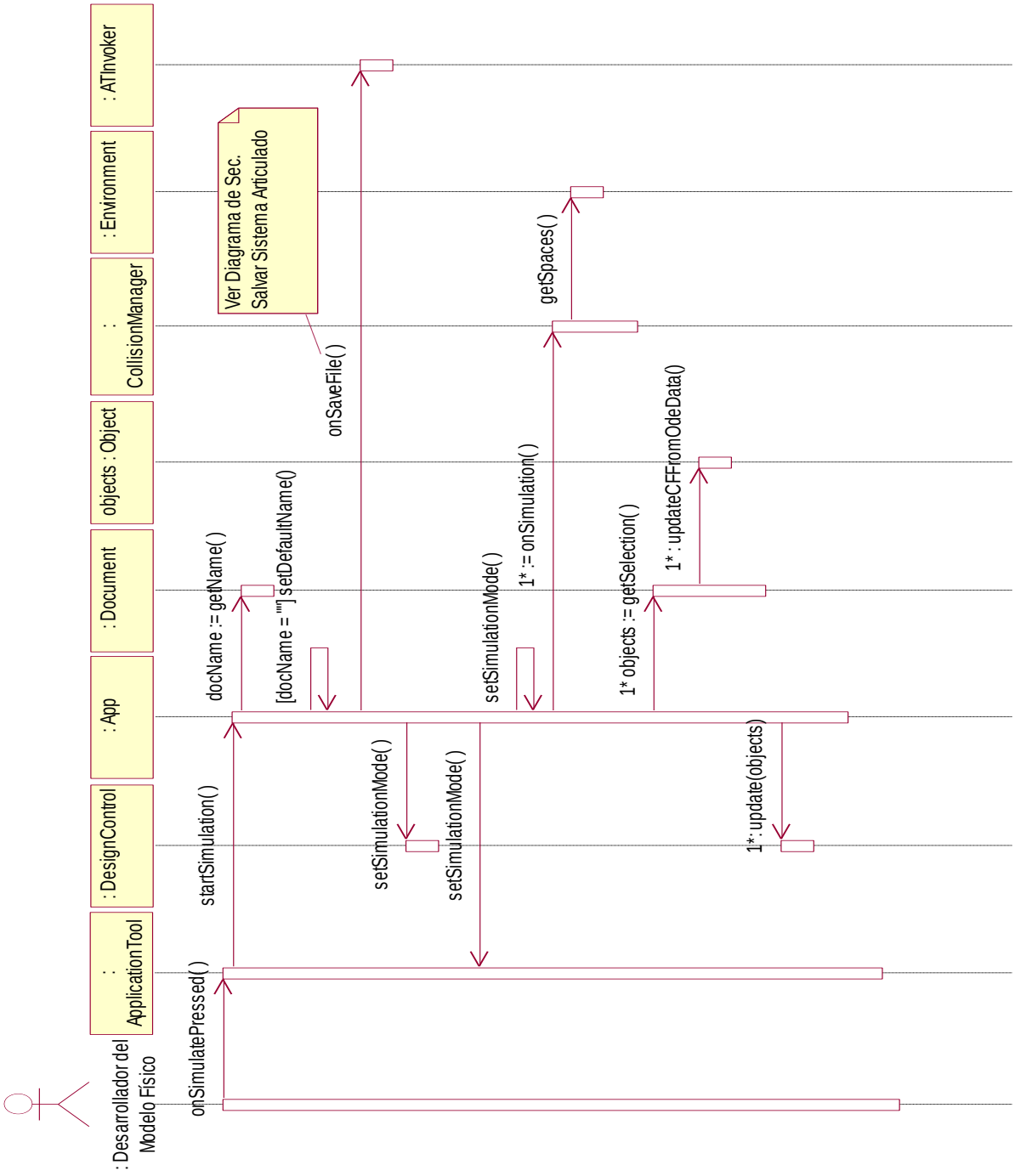
CU Crear Nuevo Documento



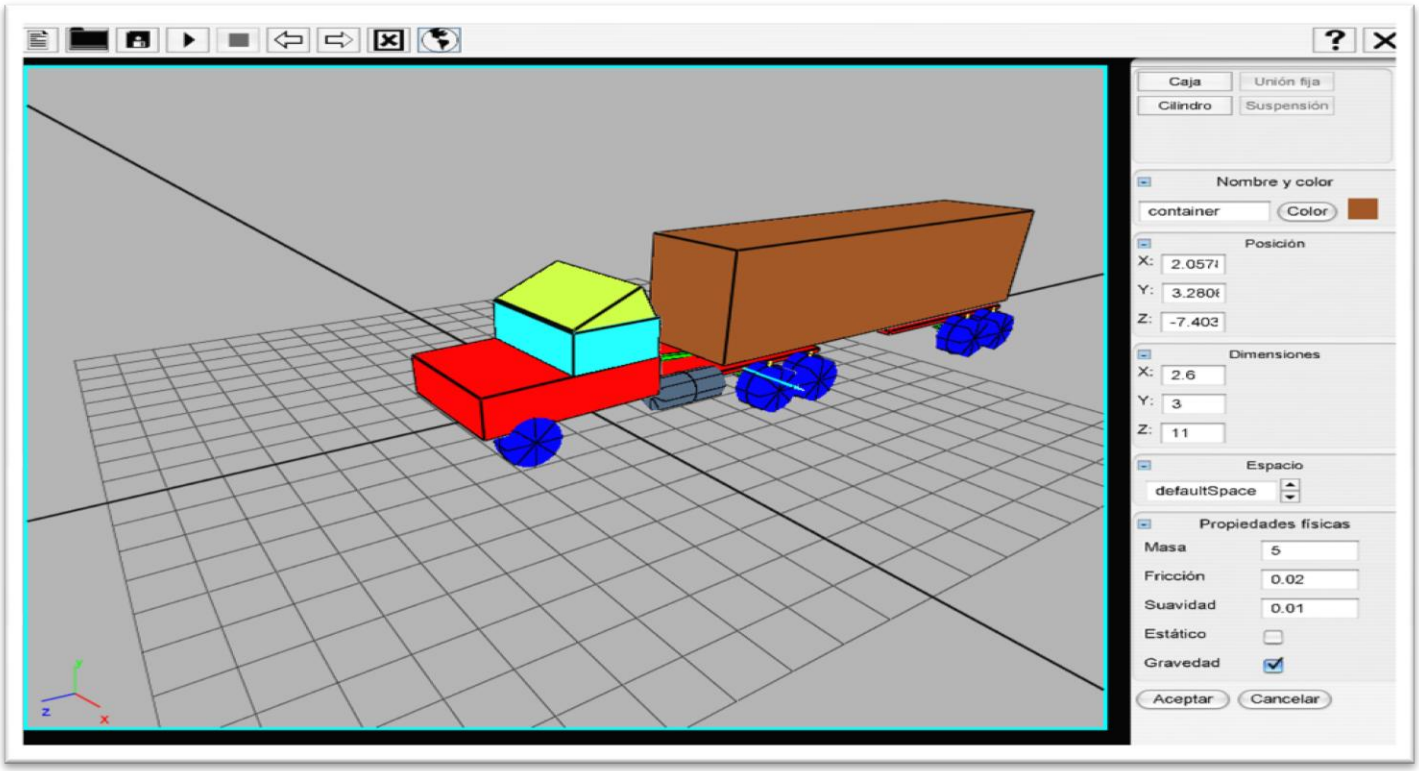
CU Salvar Sistema Articulado

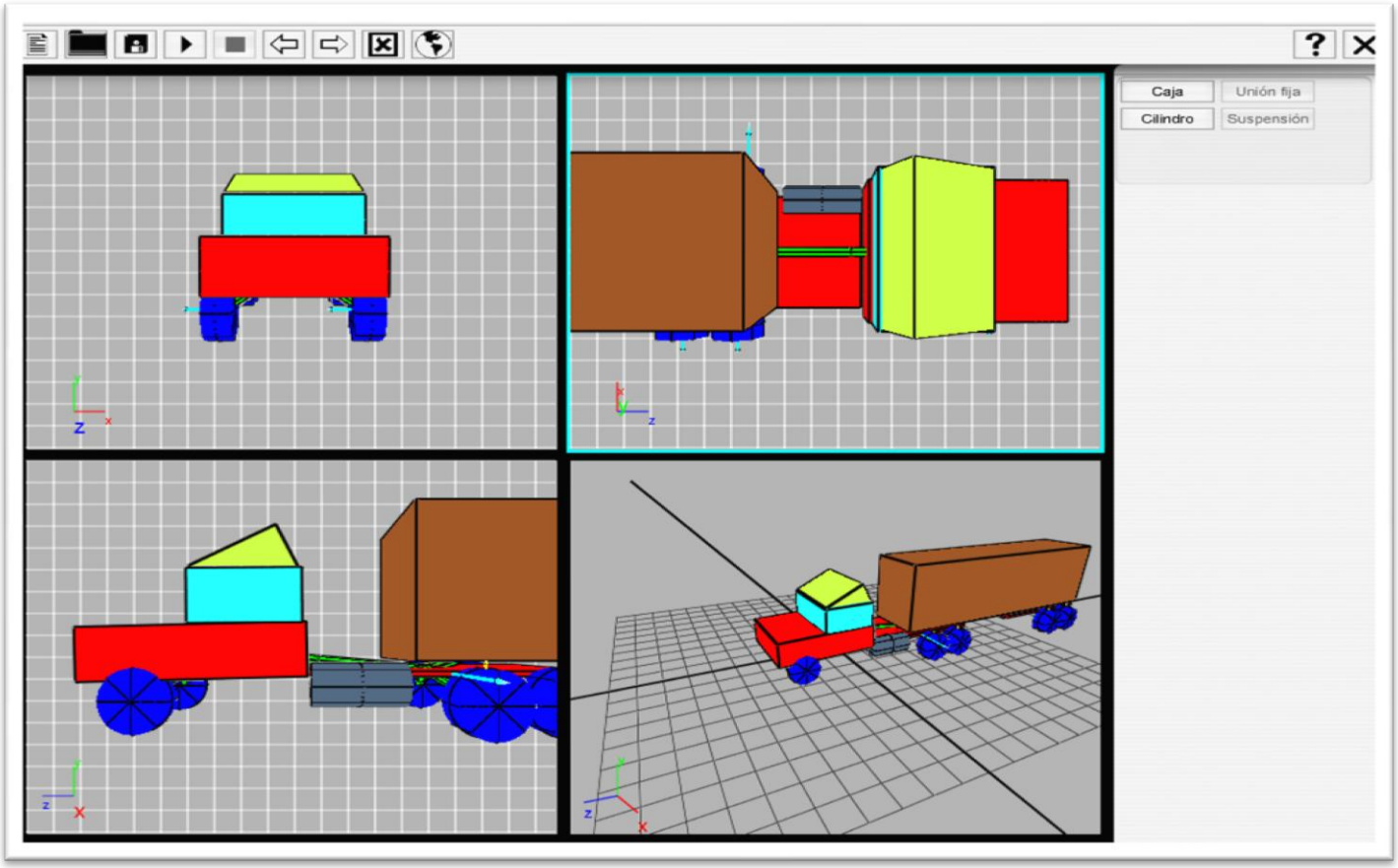


CU Simular Sistema Articulado



Anexo 8. Ejemplo de un sistema articulado





GLOSARIO DE TÉRMINOS

Application Program Interface: constituyen métodos específicos prescritos para un sistema operativo o cualquier otro programa a través del cual un programador escribe un programa de aplicación que puede formular peticiones a ese sistema operativo o a ese otro programa.

Atari**:** empresa que desarrolla, publica y distribuye juegos para la mayoría de las consolas y computadores personales. Es unos de los más grandes productores de videojuegos independientes en Estados Unidos.

Framework: conjunto de clases que contienen un diseño abstracto para soluciones a un número de problemas relacionados.

Plugin: un plugin (o plug-in, también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una funcionalidad o utilidad, generalmente muy específica.

Rendering: crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

Shader: conjunto de instrucciones gráficas destinadas para el acelerador gráfico, estas instrucciones dan el aspecto final de un objeto. Los *shaders* determinan materiales, efectos, color, luz, sombra.

Socket**:** método para la comunicación entre un programa cliente y una programa servidor en una red.