Universidad de las Ciencias Informáticas
FACULTAD 5

Título: Sistema de Toma de Decisiones Para Videojuegos

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Ever Antonio Homer Reynoso

Iriam Alberto González Boffill

Tutores: Msc. Yuniesky Coca Bergolla

Ing. Alberto Eliseo Pacheco Allende

Ciudad de la Habana

Junio, 2009

DATOS DE CONTACTO

- Msc. Yuniesky Coca Bergolla (ycoca@uci.cu)

Graduado de Licenciatura en Ciencias de la Computación en el curso 2002-2003 en la Universidad Central de Las Villas.

Se incorpora a trabajar en la Universidad de las Ciencias Informáticas donde se desempeña como Jefe de Departamento de Práctica Profesional y Realidad Virtual.

Defendió satisfactoriamente en el curso 2006-2007 su maestría en Informática Aplicada.

Obtuvo el Sello Forjadores del Futuro en el 2006 y 2008.

- Ing. Alberto Eliseo Pacheco (aepacheco@uci.cu)

Se desempeña como instructor recién graduado en la Universidad de las Ciencias Informáticas.

Agradecimientos

Agradezco a mi familia y mis amigos por hacérmelo fácil.

A mis padres por ser mis padres.

A mi ahijada .

Iriam.

Agradezco a mi familia, a mi mama, a mi abuela y mi abuelo, a mi tío y mi hermano. A todos mis amigos “los de años como me dicen aquí” y los más nuevos, principalmente a Ana Ivis por acompañarme siempre. A Elizabeth, mi novia, por soportarme tanto, y ser mi guía en los finales y el comienzo que se avecina. A Sucel por ayudarme en estos años de universidad. A todos. Ellos saben.

Ever

Dedicatoria

A aquellos que le debo todo y creen que no les debo nada.

Iriam

A los que dice Iriam y otros más.

Ever

RESUMEN

La necesidad de lograr un mayor realismo e ilusión de inteligencia en los videojuegos y simuladores producidos en el Polo Productivo de Realidad Virtual de la Facultad 5 en la Universidad de las Ciencias Informáticas, impone a los desarrolladores contar con un sistema de Inteligencia Artificial que les permita reutilizar código, minimizar el tiempo de duración del ciclo de vida del proyecto, lograr la aspirada independencia tecnológica en esta área del conocimiento para además de no depender de productos propietarios y de terceros, contribuir con la comunidad del software libre a nivel universitario, nacional e internacional. Este trabajo presenta un sistema de toma de decisiones para videojuegos el cual será un punto de partida en el desarrollo del sistema de inteligencia artificial en la Facultad. Este producto podrá ser utilizado junto con la Herramienta de Desarrollo de Sistemas de Realidad Virtual (STK[1]), o cualquier otro motor gráfico[2] en los proyectos actuales del Polo Productivo. Se implementa una versión primaria, incorporando solamente dos de las técnicas más utilizadas en la toma de decisiones dentro de la inteligencia artificial para videojuegos : la lógica difusa y las máquinas de estados finitos.

PALABRAS CLAVE

Inteligencia Artificial, Videojuegos, Realidad Virtual, Toma de Decisiones, Lógica Difusa, Máquinas de Estados Finitos.

TABLA DE CONTENIDOS

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 5

Introducción 5

1.1. Inteligencia Artificial 6

1.2. Inteligencia Artificial para Juegos 6

1.2.1. Técnicas Deterministas 7

1.2.2. Técnicas no Deterministas 8

1.2.3. El motor de IA 9

1.3. Toma de Decisiones e Inferencia 11

1.3.1. Sistemas Basados en Reglas 13

1.3.2. Lógica Difusa 14

1.3.3. Máquinas de Estados 15

1.3.4. Árboles de Decisión 17

1.3.5. Mensajes 18

1.4. Tendencias Actuales 19

1.4.1. Motores de Juegos Comerciales 19

1.4.2. Motores de Juegos Libres 19

1.4.3. Herramientas de Lógica Difusa 20

1.5. Incursiones en el país y en la Universidad. 21

1.5.1. Cuba 21

1.5.2. UCI 22

1.6. Metodologías y herramientas de desarrollo 22

1.6.1. Metodología RUP 22

1.6.2. Herramientas 23

1.7. Lenguajes de Desarrollo 24

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA 25

Introducción 25
2.1. Objeto de Estudio 26
2.1.1. Problema y situación problémica 26
2.1.2. Objeto de Automatización 27
2.1.3. Información que se maneja 27
2.1.4. Propuesta de Sistema 27
2.2. Reglas del negocio 36
2.3. Modelo del Dominio 36
2.3.1. Glosario de Términos del Modelo del Dominio 37
2.4. Especificación de requisitos de software 38
2.4.1. Requisitos Funcionales 39
2.4.2. Requisitos No Funcionales 40
2.5. Definición de los Casos de Uso del sistema 40
2.5.1. Actor del Sistema 41
2.5.2. Casos de Uso del Sistema 41
2.5.3. Diagrama de Casos de Uso del Sistema 44
2.5.4. Expansión de los casos de uso del sistema 45
CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA 52
Introducción 52
3.1. Diagramas de Secuencia 53
3.2. Diagrama de Clases del Diseño 60
3.2.1. Diagrama de Clases de Diseño por paquetes 60
3.2.2. Paquete "Módulo Lógica Difusa" 61
3.2.3. Paquete "Módulo Parser" 63
3.2.4. Paquete "Módulo Máquina de Estados" 64
3.2.5. Relación entre las clases de los paquetes. 65
3.3. Descripción de las Clases 65
3.3.1. Descripción Paquete "Módulo Lógica Difusa" 65

3.4. Diagrama de Despliegue 92
3.5. Diagramas de Componentes 92
3.5.1. Paquete "Módulo Máquina de Estados" 92
3.5.2. Paquete "Módulo Lógica Difusa" 93
3.5.3. Paquete "Módulo Parser" 94
CONCLUSIONES 95
RECOMENDACIONES 96
REFERENCIAS BIBLIOGRÁFICAS 97
BIBLIOGRAFÍA 100
ANEXOS 101
GLOSARIO 107

INTRODUCCIÓN

La proliferación de la industria del entretenimiento ha posicionado a la producción de videojuegos en una de las ramas de más adeptos en el mercado del software, tanto desarrolladores como jugadores. Los videojuegos surgen como medio de entretenimiento a principios de los 70 y en la actualidad, constituyen una de las áreas más rentables y cotizadas del mundo de la informática y las tecnologías.

La industria de los videojuegos alcanzó los 60 mil millones de dólares solamente en los Estados Unidos durante 2007/2008 y el tiempo dedicado a los juegos ha comenzado a superar el dedicado a la televisión en algunos segmentos de la población. Los presupuestos destinados al desarrollo de videojuegos de primera línea alcanzan actualmente a los presupuestos dedicados a películas animadas en el orden de 20 a 100 millones de dólares y con ganancias de 250 a 1000 millones. (1)

El desarrollo de este mercado ha traído consigo que aumente considerablemente la complejidad estructural desde el punto de vista técnico de estos productos. Se trata de un campo de estudio muy amplio dentro de la informática actual, que ha demostrado las ventajas de su aplicación en muchas disciplinas, como la medicina, la defensa o la educación. Un factor elemental en este incremento de la complejidad ha sido el empleo de técnicas de Inteligencia Artificial (en lo adelante IA) como redes neuronales artificiales, programación evolutiva, máquinas de estados y otras. Conjuntamente este aumento de complejidad ha desembocado proporcionalmente en una mayor calidad de los contenidos mostrados, haciendo que los videojuegos aumenten el interés y la adicción –no siempre en el mejor sentido de la palabra- de los jugadores.

En los videojuegos, la IA permite que el jugador crea que los personajes[I] que se encuentran en el mundo donde se desarrolla la acción tienen un comportamiento con un cierto grado de inteligencia, debido a la capacidad de estos para tomar decisiones en el entorno virtual del videojuego, permitiendo casi siempre, anticiparse a las acciones del jugador humano, obligándolo a pensar un poco más, a superarse en sus estrategias, y lo más importante en este contexto de los videojuegos, a divertirse. Implementar esta

[I] En el resto del documento se hará referencia a “personajes” - del juego - a “agentes” y “caracteres”, como una manera de expresar el mismo concepto.

inteligencia en los personajes para dotarlos de la capacidad de tomar decisiones, constituye, hoy día, uno de los pilares tecnológicos más importantes y desafiantes en el desarrollo de videojuegos.

La manera de desarrollar la IA en los videojuegos ha evolucionado desde los inicios de esta práctica. Inicialmente estas técnicas eran codificadas para cada juego y para cada personaje. Actualmente existe una creciente tendencia a tener en los motores de juegos, herramientas genéricas y reusables para la incorporación de IA. Algunas compañías privadas desarrollan estas herramientas para la creación de sus productos, con el inconveniente de que no son accesibles para la comunidad de desarrolladores independientes e imponen costosas licencias para su uso. La comunidad de software libre ha contribuido con numerosas propuestas de herramientas que automatizan diversas técnicas, con el inconveniente del poco soporte brindado por los equipos de desarrolladores y la especificidad en sus soluciones. Algunas fueron desarrolladas hace algunos años atrás y cumplían excelentemente con los paradigmas imperantes en el mercado en su momento, pero con el paso del tiempo han quedado obsoletas y no se adaptan a los requerimientos que la comunidad de desarrolladores exige, la documentación es otro requerimiento ausente en la mayoría de estos productos.

Cuba, en los últimos años ha potenciado el desarrollo de software y ha incursionado recientemente en el ámbito de la producción de videojuegos y sistemas de realidad virtual, siendo centro esencial de este desarrollo la Universidad de las Ciencias Informáticas (UCI). En esta casa de altos estudios, la Facultad 5, responsable del Polo Productivo de Realidad Virtual (en el resto del documento PPRV), ha realizado diferentes investigaciones sobre estos temas, que de forma teórica han aportado significativamente. No obstante, sus resultados prácticos han sido aislados, muchos no se han estandarizado y han desembocado en muchas soluciones que distan de contribuir a tener una visión única global de cómo podrían utilizarse juntas para elaborar un producto de realidad virtual, ya sea un simulador o un videojuego acorde a las tendencias de calidad actuales con un mínimo esfuerzo de desarrollo y en el menor tiempo posible.

En la actualidad, se hace necesario incrementar los productos con mayor calidad en el PPRV, así como el nivel de complejidad y dinamismo de los juegos que en él se promueven. El proyecto productivo “Herramienta de Desarrollo de Sistemas de Realidad Virtual” tiene una misión clave dentro de este objetivo. Se hace necesario desarrollar un módulo genérico básico para la incorporación de IA a

elementos virtuales en dicha herramienta. Y como elemento central de dicho módulo de IA es inevitable contar con un sistema de toma de decisiones que ejerza como motor central, que sea suficientemente independiente del motor gráfico y además genérico para que se pueda utilizar en cualquier tipo de juego que se desee desarrollar.

Luego, se presenta el **problema científico**, expresado en la siguiente interrogante: ¿Cómo desarrollar un sistema de toma de decisiones genérico para la incorporación de IA a proyectos del Polo Productivo de Realidad Virtual de la Facultad 5 de la UCI?

El **objeto de estudio** lo constituye la Inteligencia Artificial para videojuegos.

Como **objetivo general** se propone desarrollar un módulo genérico básico para la incorporación de IA a proyectos del Polo Productivo de Realidad Virtual de la Facultad 5 de la UCI.

Dicho objetivo se enfoca en el **campo de acción**: Técnicas y herramientas de toma de decisión para agentes inteligentes en entornos virtuales.

A continuación se presentan un grupo de tareas que permitirán satisfacer el objetivo planteado anteriormente:

1. Identificación de las diferentes aplicaciones de la Inteligencia Artificial en los videojuegos.
2. Descripción de las técnicas empleadas en el desarrollo de IA para videojuegos.
3. Evaluación de herramientas existentes en el mercado.
4. Entrevista a jefes de proyectos para capturar los requisitos principales.
5. Identificación de las técnicas a implementar en la primera versión del sistema.
6. Análisis y diseño del sistema.
7. Implementación del sistema.

Los métodos utilizados para realizar esta investigación son:

- Histórico-Lógico.
- Analítico-Sintético.

- Entrevista.
- Modelación.
- Experimento.

Con el desarrollo de este sistema se espera poder agilizar el proceso de desarrollo de videojuegos y otros productos de realidad virtual al contar con una biblioteca genérica, independiente del motor gráfico, sistema operativo y plataforma de desarrollo. Además se pretende con la realización de este trabajo, contar con una bibliografía actualizada del sistema y las técnicas empleadas, que permita la retroalimentación de los desarrolladores y la generación de conocimiento en la comunidad científica universitaria y del país.

El documento está estructurado en tres capítulos. En el primero se caracterizan las técnicas de IA aplicadas a videojuegos, especialmente las que permiten tomar decisiones, no se exponen todas, solo las más conocidas y empleadas; en el segundo capítulo se describen las soluciones técnicas, en el tercer capítulo se muestra el diseño del módulo por paquetes y la estructura física del modelo de implementación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

La intención de conseguir herramientas auxiliares en el diario batallar por la supervivencia es una constante desde que el hombre comenzó el largo camino hasta hoy. Primero se pensaba en puntas más afiladas, flechas más rectas, lanzas más aerodinámicas. Después en los materiales, los metales, la química, la electricidad, el átomo, las computadoras, y ahora se quiere algo más inteligente, algo que sin hacernos competencia, nos ayude a pensar y más aun, a divertirnos. Unos de los campos donde más se contacta con estos niveles de inteligencia, es el de los juegos de computadoras.

Desde los primeros tiempos de los videojuegos se ha mantenido una realidad en este contexto: durante el juego, el jugador humano comete generalmente un cierto desliz, se considera que se está compitiendo con la computadora, o sea, se siente el jugador en una especie de torneo en el que se debe enfrentar y ganar. Este interesante tema desde sus inicios ha sido el objeto de estudio de muchos programadores y productores de videojuegos, y ha desplegado un gran número de investigaciones con el fin de proveer a la computadora de cierta inteligencia para acercarla más al modelo de pensamiento humano. Tan noble objetivo ha propiciado la adaptación de muchas de las técnicas clásicas de Inteligencia Artificial y la creación de otras nuevas. Las tendencias de la IA aplicadas a los videojuegos y las características de estas técnicas, especialmente las que permiten a los personajes y otros elementos del juego tomar una decisión, son objeto de reflexión de este capítulo.

1.1. Inteligencia Artificial

Muchas definiciones existen sobre la Inteligencia Artificial. Si se busca el significado de inteligencia artificial en un diccionario, es probable que encuentre el siguiente planteamiento: "La capacidad de un ordenador u otro equipo para llevar a cabo las actividades que normalmente se piensa que requieren de inteligencia". Esta definición proviene del *American Heritage Dictionary* del idioma inglés, cuarta edición (Houghton Mifflin Company). Otras fuentes definen inteligencia artificial como el proceso o ciencia de crear máquinas inteligentes.

Desde otro punto de vista es conveniente pensar en la IA como el comportamiento inteligente expuesto por la máquina, o quizás el cerebro artificial detrás de un comportamiento inteligente. Pero incluso esta interpretación no es completa. Para algunas personas, el estudio de la IA no está necesariamente relacionado con el fin de crear máquinas inteligentes, sino con el propósito de obtener una mejor comprensión de la naturaleza de la inteligencia humana.

1.2. Inteligencia Artificial para Juegos

En los últimos años la industria del videojuego ha experimentado un crecimiento exponencial. Las mejoras del hardware han devenido en el incremento del poder de procesamiento a disposición de los consumidores de juegos en el hogar. En parte, como resultado de esta tendencia, la inteligencia artificial juega un papel importante en el éxito o el fracaso de un juego (2). La calidad de los gráficos y el sonido en los juegos ha mejorado de tal manera que ahora es mucho más fácil para el usuario discernir acciones cuestionables o absurdas por parte de los personajes del videojuego. Por lo tanto, reflejar un cierto nivel de inteligencia, se ha convertido en un propósito de vital importancia.

En los videojuegos, la Inteligencia Artificial permite que el jugador crea que los personajes que se encuentran en el mundo donde se desarrolla la acción tienen un comportamiento con un cierto grado de inteligencia (con más o menos acierto, dependiendo de la temática del juego). (3)

No obstante, la IA de los juegos se diferencia en algunos aspectos de la IA clásica, a no ser que la IA sea el punto más fuerte del juego, como por ejemplo si se desarrolla un juego de estrategia. En ocasiones hay que conseguir un punto intermedio entre lo inteligente que puede ser un jugador virtual y el tiempo que se

necesita para calcular su comportamiento. En la implementación de la IA para juegos en ocasiones se realizan ciertos trucos o trampas para que el sistema pueda parecer lo más "vivo" posible, pero siempre alcanzando un compromiso entre la calidad de las decisiones y el tiempo necesario para calcularlas. (3)

Otra diferencia que existe con la IA clásica es que, en ocasiones, hay que lograr que la IA sea más humana. Por ejemplo, conseguir que de vez en cuando el sistema sea capaz de fallar, aun cuando este pueda calcular la trayectoria perfecta para acertar siempre. En otro caso, el jugador se aburriría si ve que nunca es posible superar a un enemigo controlado por la IA. La calidad de la IA del juego se controla muchas veces mediante diferentes niveles de dificultad, que básicamente indican hasta qué punto se debe dotar de inteligencia a los elementos y cuántas probabilidades existen de que el comportamiento que se calcule como el mejor se lleve a cabo efectivamente. (3)

Las técnicas a utilizar dependen mucho del tipo de juego que se esté diseñando, de la importancia que se le desee dar a la IA dentro del juego, así como de la cantidad de recursos que se encuentren disponibles para ella (3). Se agrupan estas técnicas en:

- **Deterministas:** El comportamiento del agente inteligente es especificado por completo; o sea, los programadores tienen que codificar todas las acciones explícitamente, dificultando y retrasando el proceso de desarrollo del juego. Las más usadas son las máquinas de estado finito, los árboles de decisión, y los sistemas de reglas de producción.
- **No Deterministas:** El grado de incertidumbre es mayor, por lo que no se puede predecir el comportamiento que seguirá el agente. Generalmente se usan técnicas como redes neuronales, algoritmos evolutivos o redes bayesianas, que facilitan el aprendizaje y la adaptación al entorno de los elementos inteligentes. No hay que codificar explícitamente todas las posibles situaciones en el juego, ya que los elementos inteligentes pueden incluso extrapolar sus comportamientos y desarrollar otros nuevos o emergentes.

1.2.1. Técnicas Deterministas

Los desarrolladores de videojuegos han experimentado con la mayoría de las técnicas de inteligencia artificial; pero sin dudas las más sencillas, eficientes, fácil de implementar, entender y depurar son las

deterministas, por lo que han sido las más usadas en este campo. Además, el tiempo del que disponen las compañías para producir un juego es insuficiente para que los desarrolladores (más enfocados en la calidad de los gráficos) se decidan a experimentar con técnicas de inteligencia artificial no deterministas que son muy difíciles de entender, implementar y probar. (18)

De este grupo de técnicas se utilizan frecuentemente las máquinas de estado finito, que definen una serie de estados en los que puede permanecer un elemento, así como las condiciones para que se produzcan transiciones entre ellos. Muchas veces se combinan con lógica difusa, para representar en lenguaje computacional conceptos imprecisos o nociones subjetivas como “cercano/lejano”. (18)

En el juego “The Sims”, la inteligencia artificial es implementada con máquinas de estado finito difusas, además utilizan técnicas específicas de A-Life2 para simular el comportamiento de organismos vivos (en este caso, personas que viven en familia). La base de la inteligencia en este juego es su motor de comportamiento, el cual asocia acciones posibles a cada objeto. Un NPC[3] debe ser capaz de reconocer y comprender el espacio que le rodea, empleando para ello técnicas de percepción. Además debe ser capaz de buscar caminos para navegar, por lo que se le presta una especial atención a los algoritmos de búsqueda como Dijkstra o A*. Los juegos de tablero como el ajedrez y el backgammon han usado árboles de búsqueda heurística con formidables resultados. Tradicionalmente en los videojuegos se han utilizado técnicas de planificación o guiones (del inglés script) para definir la conducta de los agentes, al igual que los sistemas basados en reglas y los comportamientos grupales o de manadas. Los sistemas expertos son un tipo de sistema basado en reglas que definen el conocimiento para que los agentes autónomos se comporten de manera similar a un jugador experto. (18)

1.2.2. Técnicas no Deterministas

En la actualidad el poder computacional se ha incrementado notablemente. Existen potentes tarjetas gráficas que liberan al procesador de la máquina para que pueda encargarse de otras tareas dentro del juego, por ejemplo la física y la inteligencia artificial. Esto, unido a la demanda de los usuarios de juegos más desafiantes, al deseo de los desarrolladores de lanzar un juego que marque la diferencia, así como al mayor interés de los “académicos” de la inteligencia artificial en los videojuegos como área de experimentación relativamente barata y sin riesgos, ha posibilitado que se comiencen a aplicar más frecuentemente técnicas no deterministas. (18)

El reto actual de los desarrolladores es crear juegos novedosos, divertidos, realistas y con mayor vida útil. Un buen paso en ese sentido es lograr que los NPC aprendan, evolucionen, se adapten a nuevas situaciones y exhiban un comportamiento genuino. Estos resultados son posibles de alcanzar con métodos no deterministas que algunos desarrolladores investigan con gran interés a pesar de su complejidad. (18)

Se han alcanzado excelentes resultados en juegos que usan redes neuronales, algoritmos genéticos o métodos probabilísticos como Dirt Track Racing, Creatures, Black & White, Battlecruiser 3000AD, Fields of Battle, y Heavy Gear. Generalmente se combinan con métodos deterministas más tradicionales y estudiados, conformando una especie de sistemas híbridos. (18)

Unreal Tournament es conocido y elogiado por su inteligencia artificial. El jugador puede elegir un nivel de dificultad (desde "novato" hasta "semidiós"), y más adelante el juego implementa una opción que ajusta automáticamente la dificultad al nivel del jugador humano. (18)

1.2.3. El motor de IA

La manera de desarrollar la IA en los videojuegos ha cambiado desde su surgimiento. Inicialmente la IA era implementada para cada juego y para cada personaje. Actualmente existe una creciente tendencia a tener en los motores de juegos rutinas de IA que permiten a los personajes ser diseñados en editores de niveles[4] u otras herramientas. La estructura del motor es fija y la IA de los caracteres combina componentes apropiadamente. Por tanto construir un motor de juegos, implica desarrollar herramientas de IA que sean fácilmente rehusadas, combinadas y aplicadas apropiadamente. Para soportar esto necesitamos un motor de IA que pueda ser utilizado en múltiples juegos de cualquier género. (4)

Componentes Básicos y Diseño

Generalmente los motores de IA están compuestos -de una forma u otra- por tres sistemas básicos: toma de decisión e inferencia, percepción y navegación. (5)

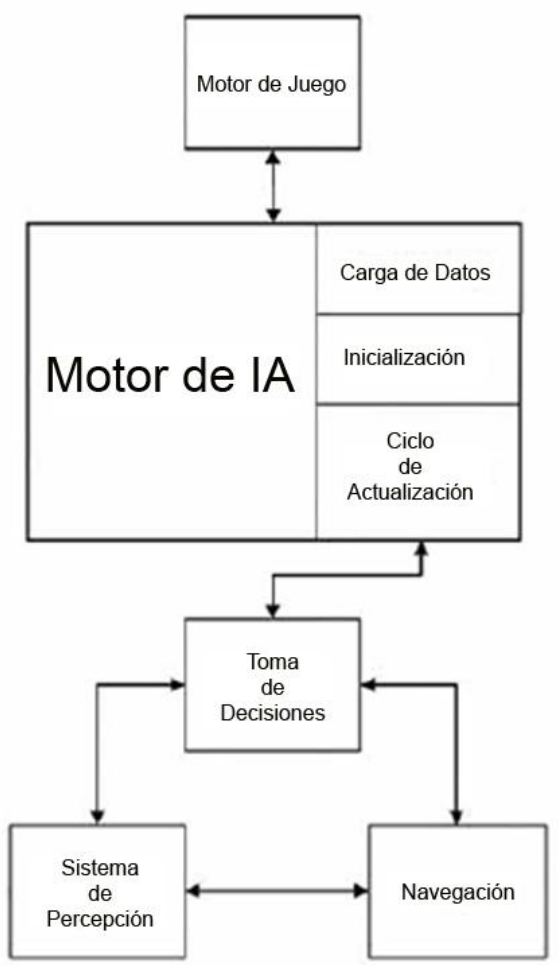

Figura 1. Diseño básico de un motor de IA. (5)

Sin embargo, Ian Millington (4) ofrece una estructura más refinada y completa de un motor de IA, aunque se mantienen los componentes esenciales a los que hace alusión Brian Schwab (5). La opinión Millington resalta que ante todo es necesario tener una infraestructura dividida en categorías: primero, un mecanismo general para manejar los comportamientos (decidiendo qué comportamiento debe ejecutarse) y un sistema que actúe de interfaz con el mundo para obtener información de él. Segundo, una interfaz estándar para controlar el movimiento y las animaciones. Y tercero, una estructura que enlace las dos anteriores.

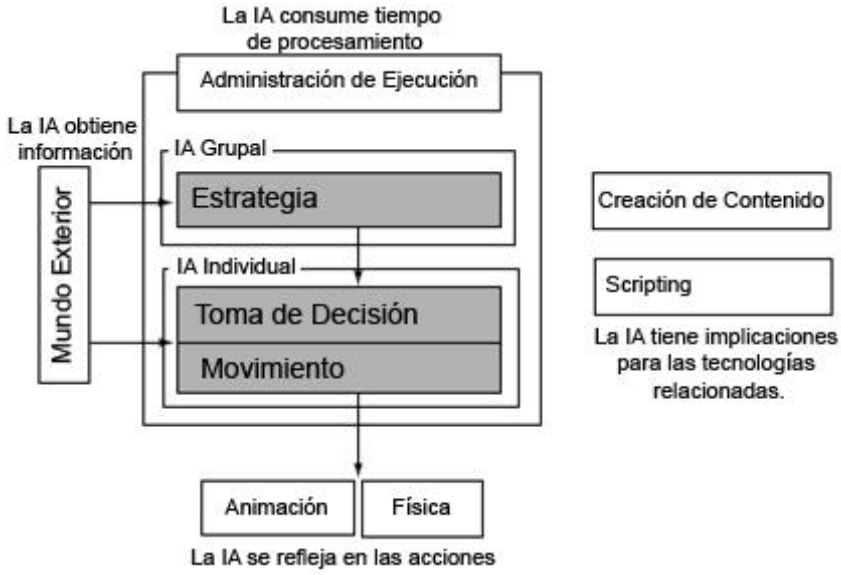

Figura 2. Motor de IA. (4)

1.3. Toma de Decisiones e Inferencia

El rol principal de la inteligencia artificial en los videojuegos es el de tomar decisiones acerca de qué debe realizar un agente inteligente durante el transcurso del juego. Del resultado de estas acciones dependerá que seamos capaces de ofrecer un reto adecuado al nivel del jugador. (4)

Inferencia se define como el acto de derivación lógica o conclusiones razonables a partir de conocimientos objetivos o premisas asumidas como ciertas. Esto significa, en términos de videojuegos, que los oponentes controlados por la Inteligencia Artificial obtienen información sobre el mundo y toman inteligentes y razonables decisiones acerca de que acción ejecutar en respuesta. (5)

En este contexto se ha desarrollado un amplio grupo de técnicas dentro de las que se encuentran, los árboles de decisión, las máquinas de estados, los sistemas basados en reglas; se adiciona también la utilización de lógica difusa y redes bayesianas que aportan al comportamiento del carácter la capacidad de trabajar con grados de incertidumbre.

En realidad, la toma de decisiones es típicamente una parte pequeña del esfuerzo que se necesita para construir la inteligencia artificial de un juego. La gran mayoría de los juegos usan un sistema de toma de decisiones sencillo: *máquinas de estados* y *árboles de decisión*. La utilización de los sistemas basados en reglas ha decrecido pero continúa siendo una técnica muy importante en esta área. (4)

En los últimos años se ha mostrado un gran interés por técnicas de toma de decisiones más sofisticadas como la *lógica difusa* y las *redes bayesianas.* (4)

La opinión de Ian Millington (4) califica el sistema de toma de decisiones como el elemento central dentro del sistema de Inteligencia Artificial de un videojuego.

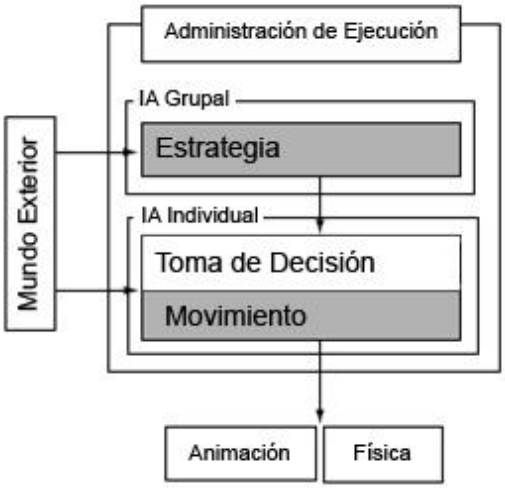

Figura 3. Módulo de Toma de Decisiones (4)

Otro planteamiento certero sobre el sistema de toma de decisiones dentro del sistema de IA lo tiene Brian Schwab (5), al calificarlo como el caballo de fuerza del sistema de IA.

El funcionamiento de un sistema de toma de decisiones es el mismo que el de cualquier sistema de gestión de información. El personaje procesa un conjunto de información que es usada para generar una acción que él desea llevar a cabo. La entrada de un sistema de toma de decisiones es el conocimiento que un personaje posee, y la salida es la acción a realizar. El conocimiento externo es la información que tiene un personaje acerca del entorno de juego: la posición de otros caracteres, el diseño del nivel, si se

ha activado un interruptor, la dirección de donde viene un sonido, y otros. El conocimiento interno es la información acerca del estado del personaje o sus procesos de pensamiento: su salud, sus metas, qué estaba haciendo un par de segundos antes, etc. (4)

Las acciones pueden tener dos componentes: una acción puede cambiar el estado externo del carácter o solamente afectar el estado interno. En las aplicaciones de juegos, los cambios en el estado interno del carácter son menos obvios, pero son decisivos en algunos algoritmos de toma de decisión. Estos cambios pueden implicar cambiar la opinión del carácter sobre el jugador, cambiando su estado emocional, o cambiando sus metas finales. (4)

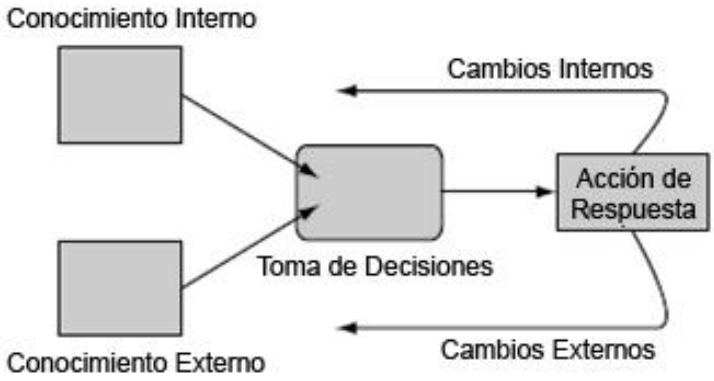

Figura 4. Funcionamiento de un módulo de toma de decisiones. (4)

En este epígrafe se presenta un conjunto de técnicas de IA aplicadas a la toma de decisiones. Todas las técnicas aquí exhibidas son aplicables a la toma de decisiones de un personaje específico o de varios personajes en conjunto.

1.3.1. Sistemas Basados en Reglas

Los sistemas de reglas son quizás el tipo de técnica de IA más utilizado en la historia de los videojuegos. En su forma más simple consisten en un conjunto de instrucciones *"If... then..."* que se utilizan para evaluar estados y tomar decisiones.

Los sistemas de reglas presentan dos ventajas principales respecto al resto de los sistemas de IA. En primer lugar, se trata de sistemas que intentan reproducir la manera de pensar y razonar de los humanos ante una situación a partir de la información que tienen de la misma. Y en segundo lugar, son muy fáciles de programar. (3)

Por otro lado, el principal problema de un sistema de reglas es que es poco escalable. Cuando se quiere implementar muchas reglas, el sistema se vuelve difícil de gestionar y el tiempo necesario para el estudio de las reglas puede provocar un efecto negativo en el rendimiento del proceso de IA. Además, los sistemas de reglas tampoco son muy efectivos cuando se dan situaciones poco comunes que no se encuentran especificadas dentro de las reglas. (3)

1.3.2. Lógica Difusa

Representa un superconjunto de la lógica convencional (Booleana) la cual ha sido extendida más allá de la binaria noción de lo verdadero o lo falso, para incluir el concepto de verdad parcial. (2)

La lógica difusa fue ideada por Lotfi Zadeh en la segunda mitad de la década del 60, permitiéndole a las computadoras interpretar términos y reglas lingüísticas de una forma similar a los humanos. Conceptos como “Lejano” o “Ligeramente” no son representados como intervalos discretos, sino como “conjuntos difusos”, consintiendo que los valores sean asignados a los conjuntos con un grado de pertenencia, mediante un proceso llamado “Fuzzificación”[5]. Usando valores fuzzificados, las computadoras están capacitadas para interpretar reglas lingüísticas que pueden ser defuzzificadas[6] y proveer un valor concreto. Esto es conocido como inferencia basada en reglas difusas, y es uno de los usos más populares de la lógica difusa.

En el dominio de los juegos el control difuso no es solamente otro método de control, es un generador de comportamiento de alta calidad. Los agentes con comportamientos inteligentes mejoran la complejidad percibida y la credibilidad dentro de un entorno virtual. Mientras los clásicos sistemas basados en reglas fracasan en muchas ocasiones en obtener sutileza, los sistemas basados en reglas difusas permiten manejar la imprecisión de los datos. Comparado con otras técnicas de inteligencia artificial, los sistemas de lógica difusa son relativamente fáciles de diseñar y de implementar. Como una extensión natural de la lógica de conjuntos, la lógica difusa está jugando un importante rol en los videojuegos.

Conjuntos Difusos

Los conjuntos difusos son una extensión de los conjuntos discretos empleados en la lógica clásica donde un elemento N pertenece o no pertenece a un conjunto dado. Esta situación puede describirse asignando un 1 a todos los elementos incluidos en el conjunto y un 0 a los no incluidos.
Sin embargo los conjuntos difusos presentan una función de pertenencia que determina el grado en que un elemento pertenece al conjunto. Por ejemplo sea U un conjunto de elementos que se denominará universo del discurso, matemáticamente un conjunto difuso F en U presenta una función de pertenencia $\mu_f = [0,1]$; donde μ_f (u) define el grado en el que u que pertenece a U es miembro de F. (28)

Reglas Difusas

Las reglas de un sistema difuso obedecen a la misma estructura de cualquier sistema basado en regla. Son del tipo *Si Antecedente Entonces Consecuencia* donde el antecedente y la consecuencia son conjuntos difusos o combinaciones de éstos, unidos mediante operadores lógicos como AND, OR y NOT. Su principal diferencia consiste en la forma en que es disparada la consecuencia, esta toma un grado de pertenencia correspondiente al grado de pertenencia resultante del antecedente.

1.3.3. Máquinas de Estados

Frecuentemente los personajes de un juego deben actuar de una forma entre varias que forman un conjunto limitado. Ellos deben llevar a cabo la misma acción hasta que algún evento o influencia externa les haga cambiar. Se puede soportar este tipo de comportamiento en un árbol de decisiones. Pero en la mayoría de los casos se usa una técnica diseñada para este tipo de situaciones: las máquinas de estados. La máquina de estados es la técnica más usada para este tipo de toma de decisiones, junto al scripting, conforman la gran mayoría de los sistemas de toma de decisión usados en los juegos actuales. (4)

Las máquinas de estados finitos (en lo adelante FSM por sus siglas en inglés) consisten en una serie de estados, un conjunto de entradas, salidas y funciones de transición de estados. Basada en una entrada y un estado inicial la función de transición calcula un nuevo estado y un conjunto de salidas. El término “finito” en las FSM se refiere a un número finito de estados que el sistema puede ocupar. (2)

Una FSM es una herramienta simple y poderosa para definir el comportamiento de un agente. Es posible dotarlas de extensiones para modelar comportamientos más complejos, tales como las máquinas de estados jerárquicas, así como las variantes no deterministas para comportamientos aleatorios. (6)

Máquinas de Estados Básicas

En una máquina de estados cada carácter ocupa un estado. Normalmente, las acciones o comportamientos están asociadas a cada estado. De esta forma, mientras el carácter se mantenga en un estado, continuará realizando la misma acción. (4)

Los estados están conectados por transiciones. Cada transición va de un estado a otro, el estado objetivo, y cada una tiene un conjunto de condiciones asociadas. Si el juego determina que las condiciones de la transición están cumplidas, entonces el carácter cambia de su estado al estado objetivo. Cuando las condiciones fueron cumplidas, se dice que ha sido activado el estado, y cuando se sigue la transición a su estado objetivo, se dice que ha sido disparado. (4)

La siguiente figura muestra una máquina de estados simple que posee tres estados: *En Guardia*, *Pelear* y *Huir*. Note que cada estado tiene su propio conjunto de transiciones. A este tipo de máquinas de estados usualmente se le conoce como máquinas de estados finitos. (4)

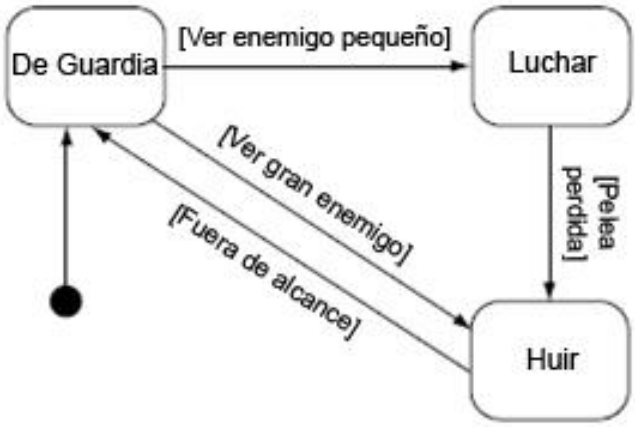

Figura 5. Máquina de Estados Simple. (4)

Máquinas de Estados Jerárquicas

En una máquina de estados jerárquica un estado puede contener una nueva máquina de estados. Por ejemplo, un estado de un agente que representa a un soldado puede ser atacar, pero el estado atacar se puede separar en cargar el fusil, apuntar y disparar. En este ejemplo el estado atacar contendría una nueva máquina de estado con los estados cargar el fusil, apuntar y disparar. (4)

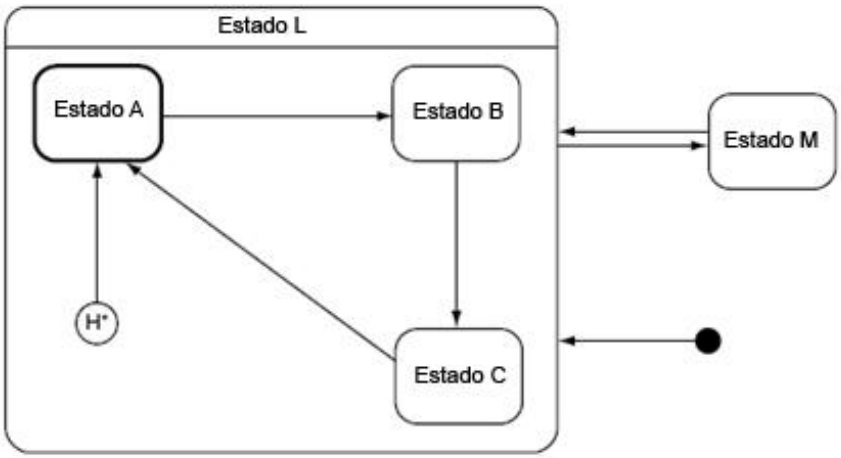

Figura 6. Máquina de estados jerárquica. (4)

Máquinas de Estados Difusas

Una máquina de estados difusa puede ser cualquier máquina de estados con algún elemento de lógica difusa. Estas pueden tener transiciones que se disparan siguiendo un modelo de lógica difusa. O puede ser que los estados contengan un grado de pertenencia a una función difusa, convirtiéndose los mismos en estados difusos.

1.3.4. Árboles de Decisión

Un árbol de decisión es una manera de relacionar una serie de entradas (usualmente tomadas del mundo del juego) a una salida (usualmente representando algo que se quiere predecir) usando una serie de reglas organizadas en estructura de árbol. (7)

Los árboles de decisión son conocidos por ser rápidos, fáciles de implementar y de entender. Son una técnica de toma de decisiones utilizadas extensivamente para controlar los caracteres o para otros tipos de toma de decisiones dentro de la inteligencia artificial para juegos como es el caso del control de animaciones. (4)

La decisión en este tipo de árboles se alcanza recorriéndolo desde la raíz hasta las hojas. El árbol de decisión suele contener cuatro tipos de elementos diferentes: (3)

- Nodos internos deterministas: contienen un test sobre la información de la situación actual que permitirá decidir qué rama elegir.
- Nodos internos probabilísticos: dependiendo de un evento aleatorio, se decidirá la siguiente rama.
- Hojas del árbol: representan el resultado que devolverá el árbol de decisión.
- Ramas del árbol: describen los posibles caminos que se extienden de acuerdo con la decisión tomada en los nodos internos.

1.3.5. Mensajes

Una de las últimas tendencias en el desarrollo de videojuegos es la programación orientada a eventos. Esto significa que cuando un evento ocurre, el acontecimiento es transmitido a los objetos en el juego de modo que puedan responder de manera adecuada. Estos eventos suelen ser enviados en forma de un paquete de datos que contiene información sobre dichos eventos: ¿cómo los enviaron?, ¿qué objetos deben responder a ellos?, ¿en qué momento fueron enviados?, etc. (22)

La razón por la cual las arquitecturas orientadas a eventos son generalmente preferidas se debe a su eficiencia. Sin un manejo de eventos, los objetos deben encuestar continuamente al entorno de juego para saber si una acción particular fue realizada. Sin embargo con el manejo de eventos, los objetos son notificados de la ocurrencia de una acción específica y actúan de forma correspondiente a este evento. (22)

Los agentes inteligentes de un juego pueden usar la misma idea con el objetivo de comunicarse entre ellos. Esta es sin duda alguna una de las técnicas más usadas en la realización de Inteligencia artificial

para videojuegos. Puede ser utilizada en la mayoría de los tipos de juegos que se conocen, desde un juego de estrategia en tiempo real hasta los juegos en primera persona. (22)

1.4. Tendencias Actuales

Los sistemas de toma de decisiones son ampliamente usados en muchos de los motores de juegos comerciales. La existencia de un sistema de este tipo en motores libres es escasa. La mayoría de los desarrolladores del mundo del software no propietario que producen videojuegos tienen que utilizar herramientas de terceros y adaptarlas a las necesidades de su producto. En este epígrafe se recoge una lista de los motores de juegos más populares que integran un sistema de toma de decisiones entre sus prestaciones de IA, y una muestra de las principales herramientas que automatizan las técnicas de lógica difusa y máquina de estados.

1.4.1. Motores de Juegos Comerciales

Unreal Engine 3: Es un framework de desarrollo de juegos muy bien avalado en el mercado. Utiliza la biblioteca gráfica OpenGL y DirectX y se ejecuta sobre los sistemas operativos Windows, Linux, MacOS y las consolas de juegos Xbox360 y PS3. Brinda la posibilidad de utilizar otras técnicas de IA: Pathfinding, Máquinas de Estados finitos y Script. (8)

Quest3D: Es un producto que provee un entorno de trabajo muy flexible para la construcción de aplicaciones 3D. Utiliza la biblioteca gráfica DirectX y se ejecuta sobre el sistema operativo Windows. Ofrece la posibilidad de utilizar otras técnicas de IA: Pathfinding, Máquinas de Estados Finitos y Script. (9)

3DGameStudio: Es un motor de juegos comercial muy cotizado en el mercado por las características que posee y por su facilidad de uso. Utiliza la biblioteca gráfica DirectX y se ejecuta sobre el sistema operativo Windows. Está desarrollado para realizar cualquier tipo de juegos. Anexa prestaciones de IA como: Pathfinding, Máquinas de Estados finitos y Script. (10)

1.4.2. Motores de Juegos Libres

Artificial Engines: Es un grupo de bibliotecas que agiliza el desarrollo de juegos con .Net y DirectX. Es un framework completamente libre. Utiliza la biblioteca gráfica DirectX y se ejecuta sobre el sistema

operativo Windows. Además, cuenta con un sistema de toma de decisiones y ofrece una librería de Pathfinding para la navegación. (11)

RealmForge: Es un motor de juegos libre, multiplataforma, escrito en C#. Se ejecuta sobre los sistemas operativos Windows, Linux, MacOS, Solaris, HP/UX, FreeBSD. Además de presentar un sistema de toma de decisiones, brinda otras técnicas de IA como Pathfinding, Máquinas de Estados finitos, Script y Redes Neuronales. (12)

Elemental Engine II: Es un motor de código abierto que ofrece todas las características necesarias para la producción y edición de un videojuego 3D. Utiliza la biblioteca gráfica DirectX y se ejecuta sobre el sistema operativo Windows, así como en las consolas de juegos PSP, Xbox360, PS2, PS3, Nintendo Wii, Nintendo DS. Presenta otras capacidades de Inteligencia Artificial como: Máquinas de estados jerárquicas y scripting con Lua. (13)

1.4.3. Herramientas de Lógica Difusa

Actualmente existen varias bibliotecas que permiten utilizar lógica difusa en las aplicaciones, específicamente videojuegos. Hay que hacer notar que la mayoría de estos productos son privativos y la licencia de uso exige grandes bonificaciones monetarias, por lo que las propuestas de la comunidad de software libre son escasas y las pocas existentes adolecen de suficiente documentación y en la mayoría de los casos el soporte para estas herramientas es deficiente volviéndose obsoletas en breve tiempo.

FFLL

Free Fuzzy Logic Library (FFLL) es una biblioteca de lógica difusa de código abierto, escrita en C++. Está optimizada para aplicaciones como los videojuegos donde la velocidad constituye un factor crítico. FFLL carga ficheros compatibles con el estándar FCL. (14)

XFuzzy 3.0

Es un entorno de desarrollo para sistemas de inferencia basados en lógica difusa. Está formado por varias herramientas que cubren las diferentes etapas del proceso de diseño de sistemas difusos, desde su descripción inicial hasta la implementación final. Sus principales características son la capacidad para

desarrollar sistemas complejos y la flexibilidad para permitir al usuario extender el conjunto de funciones disponibles. El entorno ha sido completamente programado en Java, de forma que puede ser ejecutado sobre cualquier plataforma que tenga instalado el JRE (*Java Runtime Environment*). (15)

JFuzzyLogic

Es otro paquete de herramientas de lógica difusa escrito en Java. Es compatible con el Fuzzy Control Language (FCL) especificación (IEC 1131p7). (16)

FCL

Fuzzy Control Language, es un estándar para la programación de control difuso, publicado por la International Electrotechnical Commission (IEC)[7]. Las especificaciones de la sintaxis FCL puede encontrarse en el documento IEC 61131-7. (17)

1.5. Incursiones en el país y en la Universidad.

Como se había mencionado anteriormente, en el país no existía una tendencia hacia el desarrollo de videojuegos y productos de realidad virtual por lo que no se cuenta con antecedentes establecidos de herramientas de toma de decisiones en este campo. Lo máximo que se tiene, como ya se había dicho, son investigaciones en técnicas de este tipo. En el presente epígrafe se enunciarán algunos ejemplos de empleo de lógica difusa – debido a que es la técnica en la que más se centra este trabajo - en algunos sectores del país y la universidad.

1.5.1. Cuba

Cuba ha fomentado la automatización de muchos servicios y procesos industriales que en su ejecución demandan un intenso control y supervisión, con el objetivo de tener información relevante para tomar decisiones importantes.

En la Universidad de Camagüey se desarrolló un método que permite cuantificar la información cualitativa brindada por el personal de inspección de líneas. Mediante esta información se puede determinar analíticamente la posibilidad de que la red o uno de sus elementos necesiten mantenimiento en un

período de tiempo determinado. Este procedimiento brinda la posibilidad de establecer el grado de deterioro de las líneas eléctricas utilizando la Lógica Difusa. (29)

1.5.2. UCI

En la Universidad de las Ciencias Informáticas existen antecedentes respecto al tema de la toma de decisiones en campos como las redes bayesianas, las redes neuronales y la lógica difusa, todos aplicados a videojuegos.

Andy Trujillo y Marvin Amado Márquez (18) constituyen pioneros en la utilización de las redes neuronales para definir el comportamiento de carros autónomos en un videojuego de carreras, agregando además capacidad de aprendizaje. También Yunerkis Prevot y Arianna Herrera (19) han estudiado el tema de las redes neuronales artificiales en entornos virtuales.

Tamara Martínez y Yaima Antúnez (20) abordan la lógica difusa desde la perspectiva del aprendizaje de los agentes autónomos en entornos virtuales. La comunicación entre agentes ha sido otro de los temas hacia el que se han volcado esfuerzos en esta técnica y precisamente el trabajo de diploma de Haydee Sánchez Berrillo y Lisandra Valdés Pera (21) hace un análisis de la Lógica Difusa, junto a otras técnicas, con el objetivo de brindar una propuesta adecuada para los proyectos productivos de la Facultad en esta temática.

1.6. Metodologías y herramientas de desarrollo

Durante la realización de la tesis se procedió a un estudio de cada una de las posibles metodologías y herramientas a utilizar. A continuación se presenta la metodología empleada y cada una de las herramientas, con las características distintivas que se tuvieron en cuenta para su selección.

1.6.1. Metodología RUP

La metodología de desarrollo puesta en práctica es RUP (Proceso Unificado de Desarrollo). RUP sirve de guía para realizar el análisis y diseño de la aplicación, haciendo uso de las mejores prácticas probadas en la industria, incorporando las lecciones aprendidas de cientos de líderes de la industria y miles de proyectos. (26)

Es la metodología usada por los proyectos de la Facultad por lo que existe una amplia experiencia de su uso. Además, posee una extensa documentación de referencia en la red y en soporte impreso.

RUP hace énfasis en la documentación, como elemento primordial que todo proyecto debe generar, contrario a lo que plantean otras metodologías de enfoque más ágil, esto propicia que en próximos ciclos de vida el equipo de desarrollo pueda retroalimentarse.

Un aporte fundamental del RUP al progreso del trabajo es su enfoque en los casos de uso como guía de todo el proceso de desarrollo. Los casos de uso permiten definir las funcionalidades del sistema en base a las necesidades del cliente, además de poder tener una trazabilidad de todos los artefactos generados en el ciclo de vida del proyecto.

1.6.2. Herramientas

Microsoft Visual Studio 2005

Las herramientas con la marca Visual Studio ofrecen a los desarrolladores de software mejores maneras de conseguir más invirtiendo menos esfuerzo en repeticiones y trabajos pesados: editores de código eficaces, IntelliSense, asistentes, etc. (27)

Visual Studio se ha concebido y probado para ser sistemáticamente confiable, seguro, interoperable y compatible. Ofrece una combinación inigualable de características de seguridad, escalabilidad e interoperabilidad. Aunque Visual Studio siempre incorpora particularidades vanguardistas, está diseñado para garantizar la compatibilidad con versiones anteriores siempre que sea posible. (27)

Rational Rose

Es una herramienta de modelado visual utilizada para el análisis y diseño de sistemas basados en objetos. Por ejemplo, muy útil para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto.

- Concepción y formalización del modelo.
- Construcción de los componentes.

- Transición a los usuarios
- Certificación de las distintas fases.

1.7. Lenguajes de Desarrollo

El problema planteado exige el desarrollo de una aplicación informática, por lo que es necesario escoger un lenguaje de programación con el cual generar el código fuente del sistema. Debido a la complejidad inherente a todos los sistemas actuales es imprescindible contar con un lenguaje de modelado visual para el análisis y diseño de la solución, en este apartado se caracterizaran los lenguajes escogidos para desarrollar la aplicación que proponemos para solucionar el problema existente.

En el desarrollo de la aplicación se utiliza C++ como lenguaje de programación, para la implementación de cada uno de los paquetes que conforman el diseño de la herramienta.

Para crear la documentación se utiliza UML como lenguaje de modelado, por las potencialidades descriptivas que posee y por ser un lenguaje estándar en el desarrollo de software profesional.

C++

En estos momentos, muchos de los juegos comerciales están escritos en C++, por su facilidad para obtener el control de los dispositivos de las computadoras, este lenguaje de programación sobresale en la industria de desarrollo de software por su eficiencia, su portabilidad, su rapidez de ejecución, y porque es orientado a objetos. Otra característica de este lenguaje que le hace sobresalir en esta área es su posibilidad de trabajar tanto a bajo como a alto nivel.

UML

En la opinión de los autores de este trabajo la principal ventaja de UML está en la posibilidad de capturar la idea de un sistema y comunicarla a todos los interesados en el desarrollo de la aplicación. Además la utilización de UML permite modelar el sistema concentrándose en su forma durante las primeras etapas del desarrollo sin abordar aspectos relacionados con la funcionalidad. UML constituye un estándar en la industria del software.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

El presente capítulo comienza con una explicación detallada del problema a resolver. Se proponen las soluciones específicas que darán paso al diseño, implementación e integración del sistema, con el objetivo de obtener una herramienta de fácil manipulación y eficiente rendimiento. Además se funda una visión del producto a desarrollar y se dan los primeros pasos en su concepción práctica, basándose en las necesidades y dificultades del cliente.

2.1. Objeto de Estudio

2.1.1. Problema y situación problémica

Si bien el PPRV y el área temática de Videojuegos de la Facultad 5 en la UCI cuentan con grupos de investigación y personal con experiencia en el empleo de técnicas de inteligencia artificial, se presentan varios problemas en la aplicación de estas técnicas a los diferentes juegos y simuladores que se desarrollan. Se puede observar la ausencia de una herramienta que permita realizar este trabajo de forma automática. Los diferentes esfuerzos se han enfocado en problemas específicos - muchos de los cuales quedan fuera del alcance de este trabajo- y no se ha logrado un producto general, reusable y fácil de utilizar.

La Facultad 5 comenzó la construcción de una biblioteca de inteligencia artificial, la cual después de un tiempo de prueba y aplicación en distintos proyectos ha demostrado que no es manejable y carece de muchas funcionalidades, esto provoca que los proyectos productivos al utilizarla demoren mucho tiempo en su desarrollo o desistan de su uso.

Bien se podría recurrir a las producciones internacionales pero estas distan mucho de adaptarse a nuestras necesidades, ya que las que presentan buena calidad, en la mayoría de los casos son propietarias y las alternativas libres tienen la dificultad del poco soporte brindado por los equipos de desarrolladores y la especificidad en sus soluciones

En la actualidad, se hace necesario incrementar los productos con mayor calidad en el PPRV-, así como el nivel de complejidad y dinamismo de los juegos que en el mismo se promueven. El proyecto productivo “Herramienta de Desarrollo de Sistemas de Realidad Virtual” tiene una misión clave dentro de este objetivo. Se hace necesario incorporar un módulo genérico básico –o biblioteca- para la incorporación de IA a elementos virtuales en dicha herramienta. Y como elemento central de dicho módulo de IA es inevitable contar con un sistema de toma de decisiones que ejerza como motor central.

Teniendo como problema científico: ¿Cómo desarrollar un sistema de toma de decisiones genérico para la incorporación de IA a proyectos del Polo Productivo de Realidad Virtual de la Facultad 5 de la UCI?

2.1.2. Objeto de Automatización

El objeto de automatización es un sistema de toma de decisiones para videojuegos. Esto no es más que una biblioteca de clases que se encargará de dar la posibilidad a un programador de proveer a un agente virtual autónomo de la capacidad de tomar decisiones mediante dos técnicas básicas: lógica difusa y máquinas de estados. Conjuntamente con este sistema se pretende tributar a un sistema de IA genérico de propósito general.

2.1.3. Información que se maneja

En este sistema se maneja la información referente a los archivos de configuración del modelo de lógica difusa y los estados que administran las máquinas de estados.

2.1.4. Propuesta de Sistema

El sistema propuesto en un principio, por las necesidades que planteaba el cliente contribuía a un sistema de IA general, luego de varios ajustes se optó por implementar un sistema de toma de decisiones que tributará a un futuro desarrollo de la anterior propuesta. En una primera entrega del sistema que corresponde a este ciclo de desarrollo se incluirán solamente dos técnicas entre muchas utilizadas para la toma de decisiones: la lógica difusa y las máquinas de estados, las cuales fueron descritas en el capítulo anterior.

Entre las diversas características de las técnicas a utilizar, su facilidad de implementación, la variedad de resultados que ofrecen en su ejecución en tiempo real, la similitud que presentan con el modelo de funcionamiento básico de nuestro sistema mental, la flexibilidad que brindan para su integración con otras técnicas de toma de decisiones y su popularidad en la comunidad internacional son las que han hecho posible su inclusión en esta versión del sistema.

La propuesta por tanto está compuesta por tres módulos principales, un módulo de lógica difusa que utiliza a su vez un módulo para interpretar un fichero de configuración con un modelo difuso basado en la especificación FCL, y un módulo que encapsula las máquinas de estados.

Lógica Difusa

En el capítulo anterior se hace referencia al concepto de conjunto difuso y se explica que el mismo presenta un valor conocido como grado de pertenencia que indica el grado en que un valor específico pertenece a este conjunto. Se proponen tres tipos de funciones de pertenencia utilizadas igualmente en la herramienta de referencia FFLL: Singleton, Trapecio, TrapecioS.

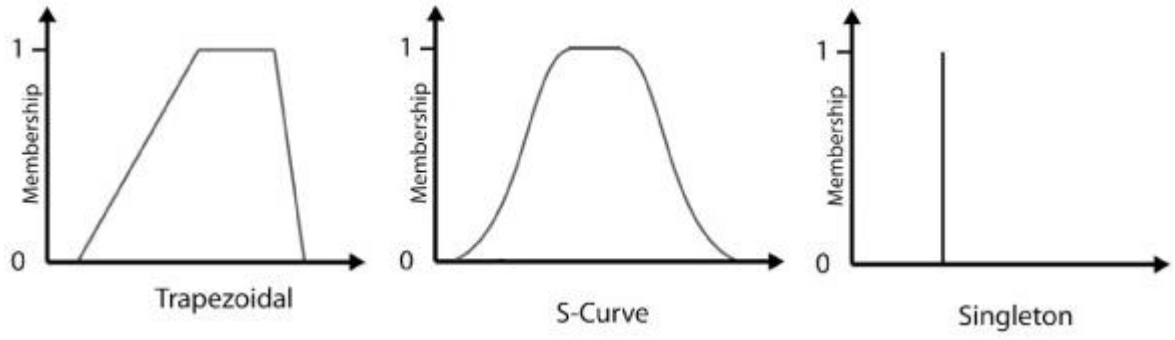

Figura 7. Tipos de Funciones de Pertenencia. (22)

Al igual que en la lógica clásica, en la lógica difusa existen operaciones entre conjuntos.

La Unión de dos conjuntos se define como:

$F_{a \cup b}(x) = Max\{F_a(x); F_b(x)\}$

La Intersección de dos conjuntos se define como:

$F_{a \cup b}(x) = Min\{F_a(x); F_b(x)\}$

El complemento o negación de un conjunto se define como:

$F_a(x)' = 1 - F_a(x)$

En la propuesta de este sistema también se ha incluido otro tipo de operación que puede aplicarse sobre conjuntos difusos conocida como aproximación o límite:

La Concentración de un conjunto difuso se define como:

$F_{Conc(a)}(x) = (F_a(x))^2$

La Dilatación de un conjunto difuso se define como:

$F_{Dilat(a)}(x) = (F_a(x))^{1/2}$

Se dice variable difusa a la composición de uno o más conjuntos difusos que representan un concepto o dominio cualitativo, por ejemplo:

Velocidad = {Lento, Medio, Rápido}

Las reglas difusas se componen de un antecedente y una consecuencia en la forma:

IF antecedente THEN consecuencia

El antecedente representa una condición y la consecuencia describe el resultado si la condición ha sido satisfecha.

El proceso de inferencia de un sistema difuso pasa por tres subprocesos fundamentales:

- Fuzzificación de las variables difusas conocidas
- Proceso de Inferencia de las Reglas
- Defuzzificación de las variables difusas resultantes

El proceso de fuzzificación consiste en determinar el grado de pertenencia de cada conjunto de la variable fuzzificada para un valor dado.

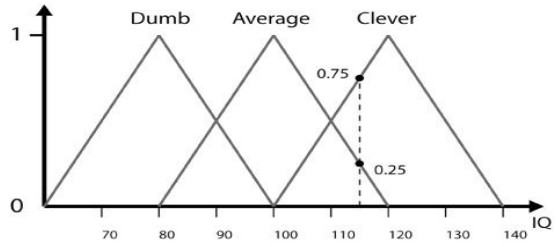

Figura 8. Proceso de fuzzificación. (22)

Por ejemplo, la fuzzificación de esta variable para un IQ=115 da como resultado un grado de pertenencia de valor 0 para el conjunto Dumb, 0.25 para Average, y 0.75 para Clever.

El proceso de inferencia pasa por los siguientes pasos:

1. Por cada regla,
    a. Calcular el grado de pertenencia del antecedente, teniendo en cuenta el grado de pertenencia de cada uno de los conjuntos difusos que lo componen y las operaciones que ocurren entre ellos.
    b. Calcular la conclusión inferida de cada regla teniendo en cuenta los valores determinados en 1ª.
2. Combinar todas las conclusiones inferidas para un conjunto difuso en una conclusión concreta.

La defuzzificación es el proceso contrario a la fuzzificación: obtener un valor concreto de la evaluación de los conjuntos difusos de una variable. Existen muchas técnicas para realizar este proceso. Esta propuesta incluye dos de las más usadas.

La técnica del Centroide o Centro de Gravedad es la más exacta pero también la más compleja de ellas. Para su desarrollo se ubica un número *n* de puntos de muestras equidistantes dentro del rango de valores de la variable, se calcula la suma de contribuciones del grado de pertenencia de cada punto de muestra al total, y se divide por la suma de los grados de pertenencia de cada punto de muestra. Obedece a la siguiente ecuación:

$$Valor\ Discreto = \frac{\sum_{s=menor\ del\ dominio}^{s=\ mayor\ del\ domino} s \times DOM(s)}{\sum_{s=menor\ del\ dominio}^{s=\ mayor\ del\ domino} DOM(s)}$$

Figura 9. Ecuación de técnica del Centroide o Centro de Gravedad. (22)

La técnica conocida como Centro de Área hace uso del máximo o *valor representativo* de los conjuntos difusos de la variable. Este término hace referencia al valor o promedio de los valores para los cuales el conjunto alcanza su mayor grado de pertenencia. Como resultado del Centro de Área se calcula la suma de los valores representativos por el grado de confianza resultante de la etapa de inferencia, y se divide entre la suma de estos grados de confianza.

$$Valor\ Discreto = \frac{\sum valor\ representativo\ x\ confianza}{\sum confianza}$$

Figura 10. Ecuación de técnica Centro de Área. (22)

En la teoría clásica proposicional existen dos importantes reglas de inferencia, el Modus Tollens y el Modus Ponens:

- El Modus Ponens o razonamiento directo puede resumirse de la siguiente forma:
  Premisa 1: “x es A”
  Premisa 2: “SI x es A, ENTONCES y es B”
  Consecuencia: “y es B”

El Modus Ponens está asociado a la implicación “A implica B” (A→B) y en términos de lógica difusa proposicional se expresa de la forma (p∩(p→q)) →q. (25)

- El Modus Tollens o razonamiento inverso puede resumirse de la siguiente forma:

Premisa 1: “y es No B”

Premisa 2: “SI x es A, ENTONCES y es B”

Consecuencia: “x es No A”

En términos de lógica difusa proposicional se expresa de la forma $(q' \cap (p \rightarrow q)) \rightarrow p'$.

El Modus Ponens es utilizado en las aplicaciones de la lógica a la ingeniería ya que conserva la relación causa-efecto mientras que el Modus Tollens apenas se utiliza. (25)

El desarrollo del módulo de lógica difusa está basado en la estructura de la herramienta FFLL (tanto esta herramienta como la especificación FCL están descritas en el capítulo anterior), de ahí que se haya definido la sintaxis de fichero de configuración basada en FCL como estándar de configuración de sistemas difusos. Esto posibilita que el programador no tenga que introducir código fuente para especificar las reglas y otros aspectos necesarios para la creación del fichero de configuración, además de poder realizar cambios en la configuración después de compilado el videojuego sin tener que recompilar nuevamente. Esta flexibilidad es esencial en el desarrollo de IA en videojuegos; además para hacer este proceso no se necesita conocer un lenguaje de programación imperativo como C++, con solo poseer conocimientos del lenguaje especificado en este documento y su modelo declarativo es suficiente, lo que posibilita delegar tareas de desarrollo de IA a personas con poco o ningún conocimiento de programación. La herramienta FFLL es compatible en cierto grado con dicho estándar, en este módulo se ha mantenido dicha compatibilidad variando en diversos aspectos.

Para la inclusión del estándar FCL se ha decidido implementar un módulo que se encargue de la interpretación del fichero de configuración. A continuación presentamos las reglas de producción del estándar FCL. Véase escrito en letras cursivas algunos elementos de estas reglas de producción que no serán usadas para la configuración de los modelos usados en esta versión del sistema.

1	Declaración_Bloque_Funciones::= ‘FUNCTION_BLOCK‘ *nombre_bloque_funciones* {declaración_variables} *{ otra_declaración_variables }* cuerpo_bloque_funciones ‘END_FUNCTION_BLOCK’

2	declaración_variables::= declaración_variables_entradas | declaración_variables_salidas
3	*otra_declaración_variables::= declaración_variables*
4	cuerpo_bloque_funciones::= {bloque_fuzzificación} { bloque_defuzzificación } {bloque_reglas} *{bloque_opciones}*
5	bloque_fuzzificación::= 'FUZZIFY' variable_nombre {término_linguístico} 'END_FUZZIFY'
6	bloque_defuzzificación::= 'DEFUZZIFY' f_ variable_nombre *{ término_linguístico }* método_defuzzificación *valor_por_defecto* *[rango]* 'END_FUZZIFY'
7	bloque_reglas::= 'RULEBLOCK' bloque_reglas_nombre *definición_operador* *[método_activación]* método_acumulación {regla} 'END_RULEBLOCK'
8	*bloque_opciones::= 'OPTION'* *cualquier parámetro específico* *'END_OPTION'*
9	término_linguístico::= 'TERM' término_nombre ':=' función_membrecía ';'
10	función_membrecía::= singleton | puntos
11	singleton ::= número_literal
12	puntos::= {'(' número_literal | *variable_nombre* ',' número_literal ')'}
13	método_defuzzificación::= 'METHOD' ':' 'COG' | *'COGS'* | 'COA' | *'LM'* | *'RM'* | *MOM* ';'
14	*valor_por_defecto::= 'DEFAULT' ':=' número_literal* | *'NC' ';'*

15	*rango::= 'RANGE' ':=' '('número_literal '..' número_literal ')' ';'*
16	*definición_operador ::= ('OR' ':' 'MAX'* | *'ASUM'* | *'BSUM')* | *('AND' ':' 'MIN'* | *'PROD'* | *'BDIF') ';'*
17	*método_activación::= 'ACT' ':' 'PROD'* | *'MIN' ';'*
18	método_acumulación::= 'ACCU' ':' 'MAX' | *'BSUM'* | *'NSUM'* ';'
19*	*regla ::= 'RULE' entero_literal ':'* *'IF' condición 'THEN' conclusión [WITH facto_de_peso] ';'*
20*	*condición::= (subcondición* | *variable_nombre) {AND'* | *'OR' (subcondición* | *variable_nombre)}*
21*	*subcondición ::= ('NOT' '('variable_nombre 'IS' ['NOT']) término_nombre ')')* | *(variable_nombre 'IS' ['NOT'] término_nombre)*
22*	*conclusión ::= { (variable_nombre* | *(variable_nombre 'IS' término_nombre)) ',}*
23	*facto_de_peso::= variable* | *número_literal*
24	*nombre_bloque_funciones::= identificador*
25	bloque_reglas_nombre::= identificador
26	término_nombre::= identificador
27	variable_nombre ::= identificador
28	número_literal::= entero_literal | real_literal
29	letra ::= 'A' | 'B' | <...> | 'Z' | 'a' | 'b' | <...> | 'z'
30	dígito ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
31	identificador ::= (letra | ('_' (letra | dígito))) {['_'] (letra | dígito)}
32	declaración_variables_entradas::= 'VAR_INPUT' *['RETAIN'* | *'NON_RETAIN']* entrada_declaración ';' { entrada_declaración ';'} 'END_VAR'
33	entrada_declaración::= inicio_decl_var | *limite_declaración*
34	inicio_decl_var::= inicio_decl_var1 | *arreglo_inicio_decl_var* | *estructura_inicio_decl_var* | *fb_nombre_decl* | *string_var_decl*
35	inicio_decl_var1::= var1_lista ':' *(inicio_espec_simple* | *subrango_inicio_espec* | *enum_inicio_espec)*
36	var1_lista::= variable_nombre *{',' variable_nombre }*

37	*array_var_init_decl ::= var1_lista ':' inicio_arreglo_espec*
38	declaración_variables_salidas::= 'VAR_OUTPUT' *['RETAIN'* | *'NON_RETAIN']* inicio_decl_var ';' { inicio_decl_var ';'} 'END_VAR'
39	tipo_real_nombre ::= 'REAL' | *'LREAL'*
40	tipo_número_nombre ::= *tipo_entero_nombre* | tipo_real_nombre
41	tipo_elemental_nombre::= tipo_numero_nombre | *tipo_fecha_nombre* | *tipo_bit_string_nombre* | *'STRING'* | *'WSTRING'* | *'TIME'*
42	tipo_simple_nombre ::= identificador
43	tipo_simple_declaración::= tipo_simple_nombre ':' inicio_espec_simple
44	especificación_simple ::= tipo_elemental_nombre | *tipo_simple_nombre*
45	inicio_espec_simple:= especificación_simple *[':=' constante]*

Tabla 1. Reglas de producción del estándar FCL. (17)

En el caso de las reglas de producción del 19, 20, 21 y 22 que han sido marcadas con un asterisco es preciso aclarar que a las mismas se les han hecho algunas modificaciones para darle más posibilidades al sistema en la creación de las reglas difusas, aplicando estas reglas de producción.

19	regla ::= ‘RULE’ entero_literal ‘:’ ‘IF’ expresión ‘THEN’ término_ref ‘;’
20	expresión::= (expresión ‘AND’ expresión) | (expresión ‘OR’ expresión) | (‘NOT’ expresión) | (‘(’expresión’)’ | límite | término_ref)
21	límite ::= (‘VERY’ término_ref) | (‘FAERLY’ término_ref)
22	término_ref ::= (término_nombre | (variable_nombre ‘IS’ término_nombre))

Tabla 2. Reglas de producción incorporadas.

Esta modificación permite el uso de operaciones dentro de operaciones, por ejemplo una expresión podría ser: NOT (a AND b). Además se le incorpora al compilador un árbol de precedencia de operadores para el trabajo con este tipo de expresiones.

Máquina de estados finitos

Las máquinas de estados están diseñadas de manera que cada estado es encapsulado en una clase independiente.

Una clase se encarga de administrar todos los estados existentes y sus transiciones.

El módulo de máquina de estados está basado principalmente en la aplicación del patrón GOF[8] State[9] con el objetivo de contribuir a la reusabilidad del módulo. La referencia esencial en este caso es Mack Buckland (22).

2.2. Reglas del negocio

El programador tiene que tener conocimientos básicos de programación orientada a objetos y del lenguaje de programación C++.

El programador debe poseer conocimientos de la sintaxis del fichero de configuración, la misma ha sido especificada anteriormente.

Toda entidad de juego que haga uso del módulo de máquina de estados debe tener una relación de generalización-especificación con la clase BasicGameEntity y esta entidad debe ser registrada en la clase EntityManager, la cual ha sido referenciada de forma estática con el objeto EntityMgr; así como todas las clases que representen estados deben tener el mismo tipo de relación con la clase State.

Si se hace uso del sistema de mensajes del módulo de máquina de estados se debe ejecutar la función Update() del la clase FrameCounter en cada actualización del mundo del juego. Esta clase ha sido referenciada de forma estática con el objeto TickCounter.

2.3. Modelo del Dominio

El modelo del dominio constituye un modelo conceptual del entorno de la aplicación o módulo a desarrollar. En el siguiente diagrama se recogen los principales conceptos del contexto en el que se ejecuta el sistema, así como las relaciones existentes entre ellos. Con el propósito de lograr simplicidad no se incluyen los modelos conceptuales de las técnicas específicas del sistema.

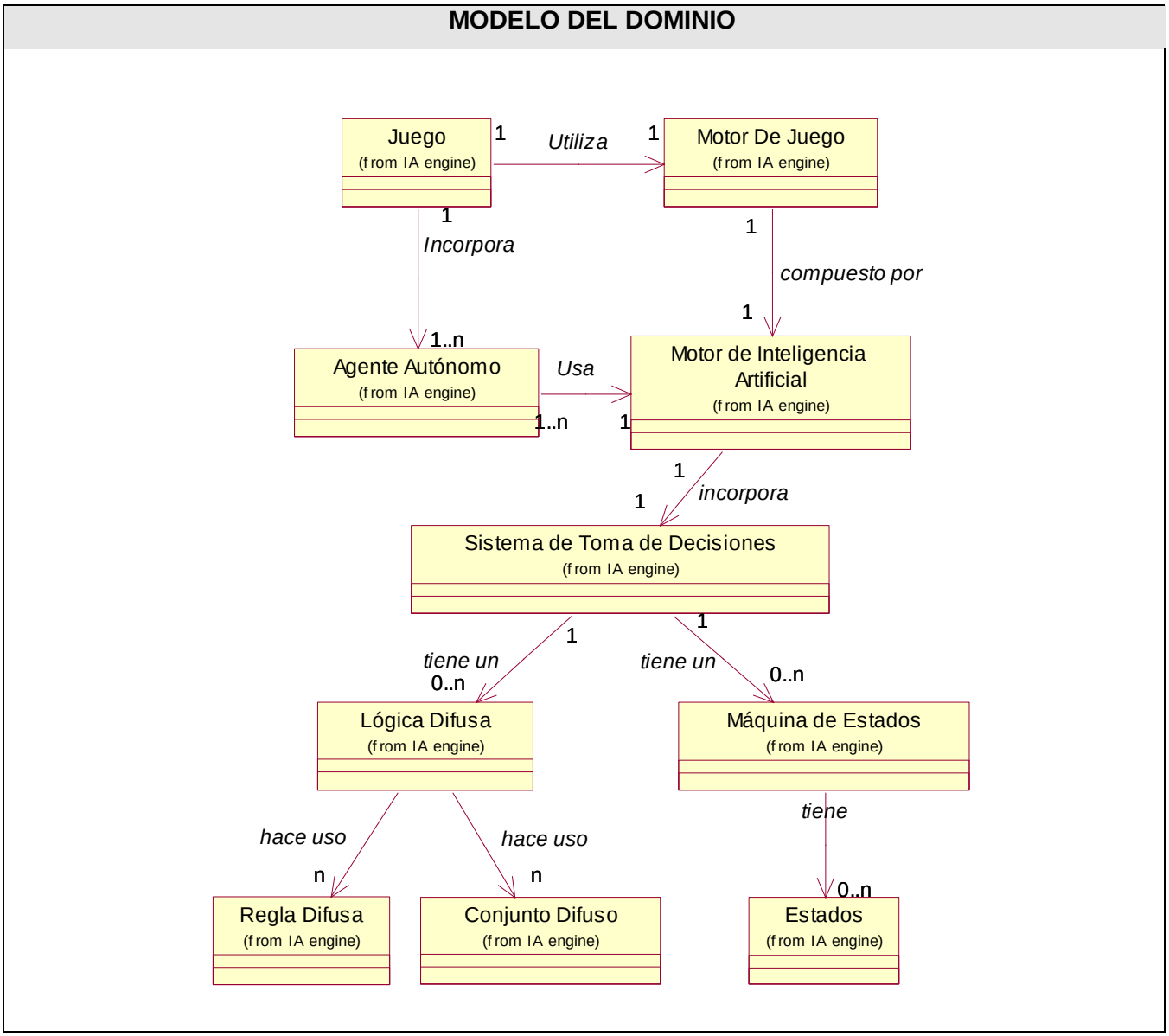

Figura 11. Modelo Del Dominio

2.3.1. Glosario de Términos del Modelo del Dominio

- **Videojuego:** Programa electrónico que genera elementos para todo tipo de actividades lúdicas con reglas determinadas y cuya interfaz se sustenta en imágenes, luces y sonidos transmitidos por dispositivos diseñados para tal efecto.(23)
- **Motor de Juego (Game Engine):** Software diseñado para la creación y desarrollo de videojuegos.
- **Agente Autónomo:** Un agente autónomo es un sistema situado en un entorno del que forma parte y sobre el cual actúa, a través del tiempo, persiguiendo sus propios objetivos de forma que afecte lo que siente en el futuro.(24)
- **Motor de Inteligencia Artificial:** (Ver epígrafe 1.2.3).
- **Sistema de Toma de Decisiones:** (Ver epígrafe 1.3).
- **Lógica Difusa:** Es la lógica aplicada a conceptos que pueden tomar un valor cualquiera de veracidad dentro de un conjunto de valores que oscilan entre dos extremos, la verdad absoluta y la falsedad total. (25)
- **Regla Difusa:** Se llaman reglas difusas al conjunto de proposiciones IF-THEN que modelan el problema que se quiere resolver. (25)
- **Conjunto Difuso:** Generalización del concepto clásico de conjuntos (25). Contempla la pertenencia parcial de un elemento al conjunto.
- **Máquina de Estados:** Es una estructura de datos que contiene tres elementos fundamentales: todos los estados inherentes a esta máquina, algunas condiciones de entrada, y una función de transición que crea una línea de conectividad entre estos estados.(5)
- **Estado**: En este caso un estado es un grupo de acciones que debe realizar un objeto que haga uso de la máquina de estados.

2.4. Especificación de requisitos de software

La captura de requisitos constituye el primer paso para el desarrollo de cualquier sistema, sea de software o no. Los requisitos especifican lo que se desea de la aplicación, tanto las capacidades como las cualidades. Muchas metodologías abogan por la obtención de requisitos mediante el modelado de los procesos que intervienen en el entorno de negocio. Debido a la deficiencia del problema a tratar para establecer los límites de los procesos claramente, así como los roles que en ellos se juegan, se opta por derivar los requisitos, tanto funcionales como no funcionales de la descripción de los objetos del dominio vistos en la sección anterior.

Dependencias y relaciones con otro software

El resultado de este trabajo es un sistema de toma de decisiones, desarrollado para ser integrado a un sistema genérico de IA. Esta dependencia no es fuerte, el sistema está diseñado para ofrecer una interfaz bien definida que le brinde a cualquier sistema externo los comportamientos a los cuales tiene acceso, escondiendo los detalles de implementación y elevando la interoperabilidad entre los sistemas, tributando a un bajo acoplamiento y una baja cohesión.

El sistema no está desarrollado para ser usado con algún motor gráfico en particular.

2.4.1. Requisitos Funcionales

Teniendo en cuenta las necesidades de los futuros usuarios y la descripción de cómo debe funcionar el sistema, se pueden inferir los requerimientos funcionales siguientes.

El Sistema debe permitir

1. Crear nuevo modelo difuso.
    1.1. Adicionar Variables.
    1.2. Definir Conjuntos.
    1.3. Definir Método De Defuzzificación.
    1.4. Definir Reglas.
2. Cargar fichero de configuración del modelo difuso.
    2.1. Compilar el fichero.
    2.2. Generar fichero con información detectada durante el proceso de carga del modelo difuso.
3. Fuzzificar variable.
4. Desarrollar inferencia.
5. Defuzzificar.
6. Devolver variable de salida.

7. Adicionar nuevo estado.
8. Cambiar de estados.
9. Actualizar máquina de estados.
10. Enviar Mensajes.
11. Enviar Mensaje Retardados.
12. Adicionar Entidad
13. Eliminar Entidad

2.4.2. Requisitos No Funcionales

- **Soporte**

  Deberá ejecutarse tanto en Sistemas Operativos Windows como en entornos Unix.
- **Usabilidad**

  El sistema no brinda interfaz gráfica, en su lugar debe ofrecer una interfaz interna con la aplicación que la utiliza, con el objetivo de que sea de fácil uso para los programadores.
- **Legales**

  Se regirá por las normas ISO 9000.
- **Restricciones de diseño e implementación**

  El sistema será implementado en la herramienta Visual Studio 2005, utilizando el lenguaje de programación C++, bajo el paradigma de programación orientada a objetos.

2.5. Definición de los Casos de Uso del sistema

A continuación se reconocen y se detallan los casos de uso a partir de los requisitos funcionales anteriormente especificados, se describen también a los actores que inician los respectivos casos de uso. Los casos de uso son un elemento esencial en el proceso de desarrollo, ayudan a entender el sistema desde el punto de vista del usuario y en el caso de RUP, reviste vital importancia por su condición de guiar el proceso de desarrollo.

2.5.1. Actor del Sistema

Actor	Justificación
Aplicación Final.	Es la aplicación, módulo o sistema externo que hace uso del sistema de toma de decisiones y se beneficia de sus resultados.

Tabla 3. Justificación de Actores

2.5.2. Casos de Uso del Sistema

CU-1	Cargar Modelo.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica cargar un fichero desde un directorio físico y crear un modelo difuso con la información que este contiene, y termina cuando el modelo se crea satisfactoriamente.
Referencia	RF.1, RF.2

Tabla 4. Cargar Modelo

CU-2	Crear Modelo.
Actor	Aplicación Final.
Descripción	El caso de uso es iniciado por el caso de uso Cargar Modelo después de que el fichero ha sido correctamente compilado se procede a la creación del modelo.
Referencia	RF.1, RF.1.1, RF.1.2, RF.1.3, RF.1.4

Tabla 5. Crear Modelo.

CU-3	Desarrollar Inferencia.
Actor	Aplicación Final.

Descripción	El caso de uso se inicia cuando la aplicación final indica realizar el proceso de inferencia de las reglas que contiene el modelo y finaliza cuando esta obtiene un resultado final.
Referencia	RF.4, RF.5, RF.6

Tabla 6. Desarrollar Inferencia.

CU-4	Fuzzificar Variable.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica fuzzificar una variable y termina cuando la acción es llevada a cabo.
Referencia	RF.3

Tabla 7. Fuzzificar Variable

CU-5	Cambiar Estado.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica cambiar el estado actual de los personajes a otro estado existente.
Referencia	RF.8

Tabla 8. Cambiar Estado

CU-6	Actualizar Máquina de Estado.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica actualizar la máquina de estados y termina cuando la acción es llevada a cabo.
Referencia	RF.9

Tabla 9. Actualizar Máquina de Estado

CU-7	Enviar Mensaje.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica enviar un mensaje a uno o varios agentes y termina cuando el mensaje es depositado en el buzón.
Referencia	RF.10

Tabla 10. Enviar Mensaje.

CU-8	Enviar Mensaje Retardados.
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica enviar mensajes existentes en el buzón que tengan tiempo de retardo.
Referencia	RF.11

Tabla 11. Enviar Mensaje de Retardo.

CU-9	Gestionar Entidad
Actor	Aplicación Final.
Descripción	El caso de uso se inicia cuando la aplicación final indica agregar o eliminar una entidad en el administrador de entidades y culmina cuando esta acción ha sido llevada a cabo.
Referencia	RF.12, RF.13

Tabla 12. Adicionar Entidad.

2.5.3. Diagrama de Casos de Uso del Sistema

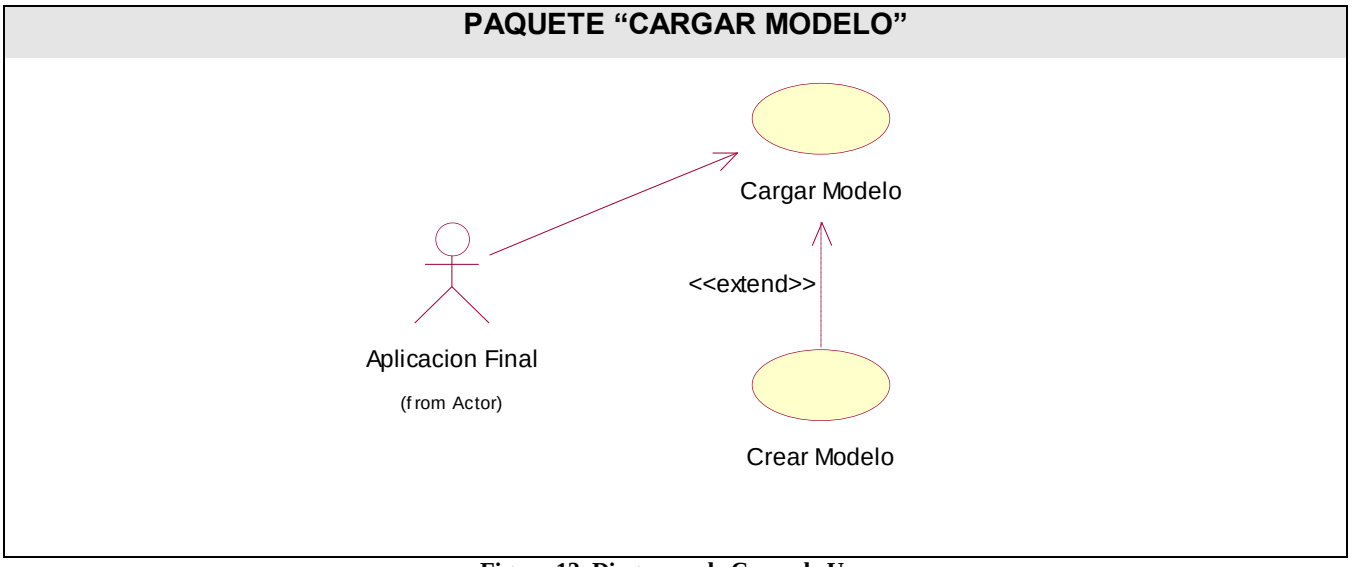

Figura 12. Diagrama de Casos de Uso.

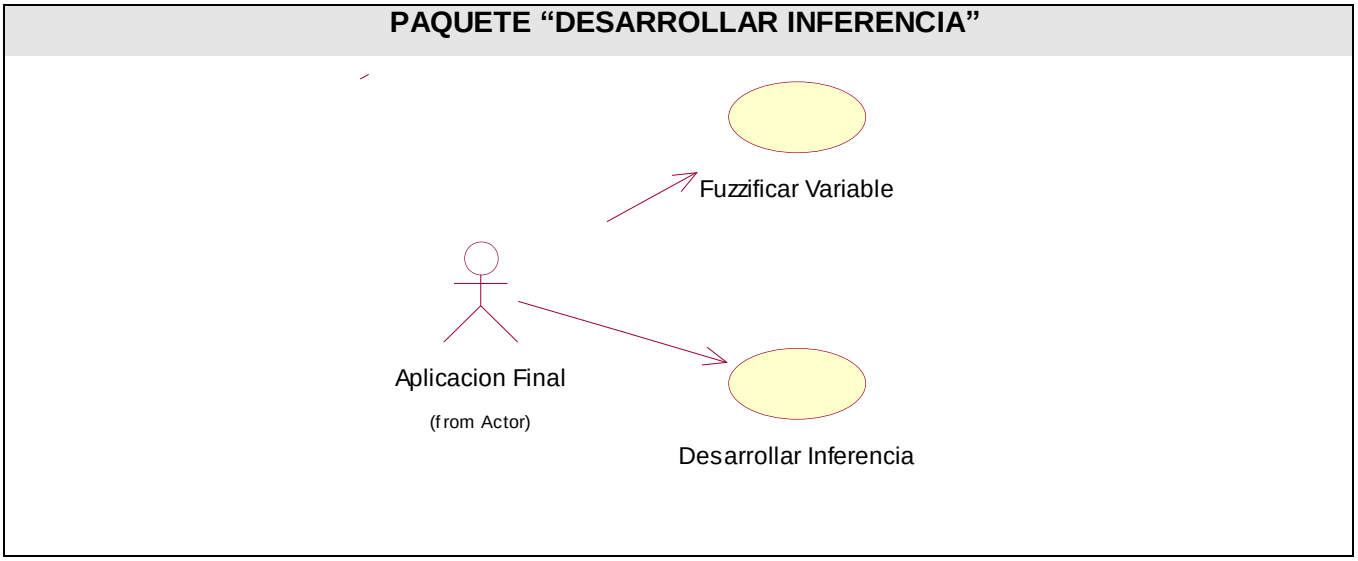

Figura 13. Diagrama de Casos de Uso.

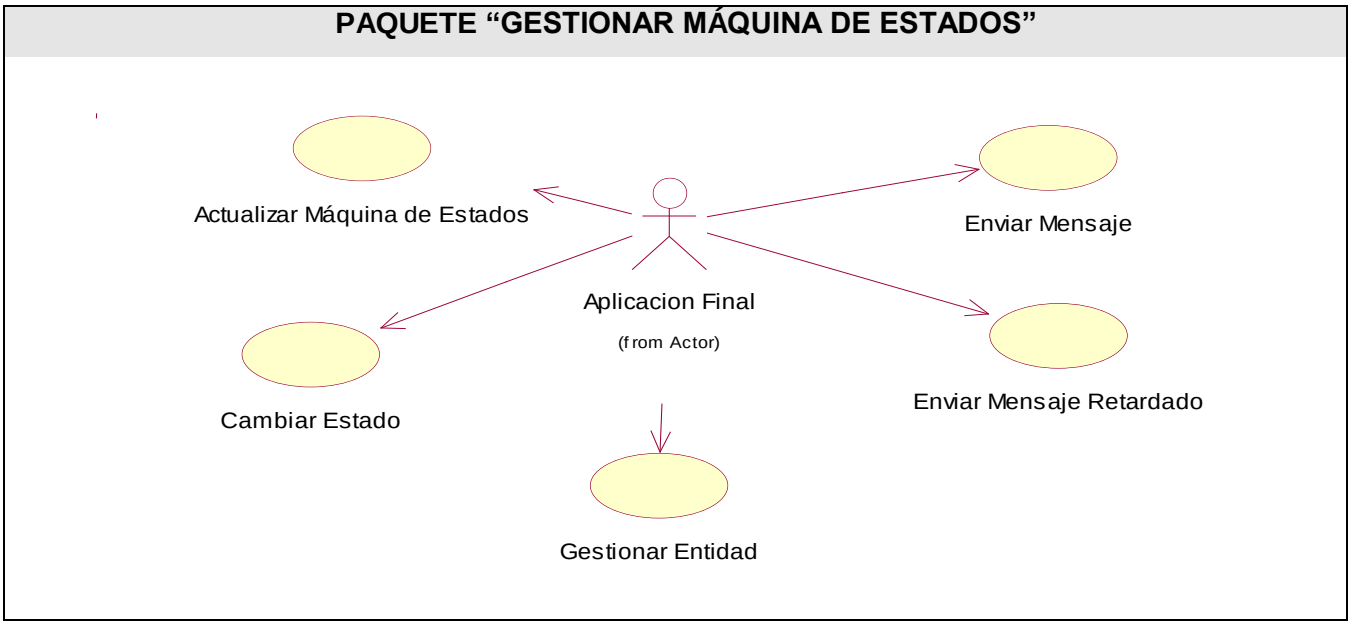

Figura 14. Diagrama de Casos de Uso.

2.5.4. Expansión de los casos de uso del sistema

Caso de uso	
CU-1	Cargar Modelo.
Propósito	Permitir cargar el fichero con la información del modelo difuso.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica cargar un fichero desde un directorio físico y crear un modelo difuso con la información que este contiene, y termina cuando el modelo se crea satisfactoriamente o se genera el reporte de errores.	
Referencias	RF.2, RF.2.1, RF.2.2
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso comienza cuando la aplicación final indica cargar un modelo.	2- El sistema carga un fichero desde una dirección física y procede a la compilación del mismo. 3- Si el fichero no contiene errores, se crea el

	modelo.
Precondiciones	Debe haber un fichero creado con la información del modelo.
Postcondiciones	Se crea el modelo.
Flujo Alternativo	
	3- Si el fichero contiene errores, se crea un fichero con un reporte de los errores contenidos en el fichero.

Tabla 13. Descripción del caso de uso Cargar Modelo

Caso de uso	
CU-2	Crear Modelo.
Propósito	Crear un modelo de lógica difusa.
Actores: Aplicación Final.	
Resumen: El caso de uso es iniciado por el caso de uso Cargar Modelo. Después de que el fichero ha sido correctamente compilado se procede a la creación del modelo.	
Referencias	RF.1, RF.1.2, RF.1.3, RF.1.4
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso comienza cuando la aplicación final indica cargar un modelo y se procede a la creación del mismo.	2- Se crea un nuevo modelo donde se definen las variables, los términos de las mismas, las reglas del modelo y el método de defuzzificación que será usado. 3- Se adiciona al administrador de modelos.
Precondiciones	Se debe haber compilado satisfactoriamente la configuración del modelo.
Postcondiciones	Se crea el modelo y se adiciona al administrador de modelos.

Tabla 14. Descripción del caso de uso Crear Modelo

Caso de uso

CU-3	Desarrollar Inferencia.
Propósito	Realizar el proceso de inferencia de las reglas que contiene el modelo.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica realizar el proceso de inferencia de las reglas que contiene el modelo y finaliza cuando esta obtiene un resultado final.	
Referencias	RF.4, RF.5, RF.6

Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación indica realizar el proceso de inferencia. 3- La aplicación solicita la variable de salida.	2- El sistema realiza la inferencia de las reglas. 4- El sistema devuelve el valor de la variable.

Precondiciones	-
Postcondiciones	Se realiza el proceso de inferencia y se devuelve un resultado final.

Flujo Alternativo	
	4- Si la variable solicitada por la aplicación no existe, se lanza una excepción.

Tabla 15. Descripción del caso de uso Desarrollar Inferencia.

Caso de uso	
CU-4	Fuzzificar Variable.
Propósito	Definir el grado de pertenencia de los conjuntos de la variable para un valor dado.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica fuzzificar una variable y termina cuando la acción es llevada a cabo.	
Referencias	RF.3

Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso si inicia cuando la aplicación final indica fuzzificar una variable.	2- El sistema verifica que la variable exista. 3- El sistema verifica que el valor de fuzzificacion este en el rango de la variable. 4- El sistema fuzzifica la variable.
Precondiciones	-
Postcondiciones	Se fuzzifica la variable.
Flujo Alternativo	
	2- Si la variable no existe, se lanza una excepción. 3- Si el valor está fuera de rango, se lanza una excepción.

Tabla 16. Descripción del caso de uso Fuzzificar Variable.

Caso de uso	
CU-5	Cambiar Estado.
Propósito	Cambiar de un estado actual a otro estado existente.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica cambiar el estado actual de los personajes a otro estado existente y termina cuando la acción es llevada a cabo.	
Referencias	RF.8
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación indica cambiar de estado.	2- El sistema cambia de estado.
Precondiciones	-
Postcondiciones	Se cambia de estado.
Flujo Alternativo	

	2- Si el estado no existe se lanza una excepción.

Tabla 17. Descripción del caso de uso Cambiar Estado.

Caso de uso		
CU-6	Actualizar Máquina de Estados	
Propósito	Según condiciones especificas hacer las transiciones de estados correspondientes	
Actores: Aplicación Final (inicia).		
Resumen: El caso de uso se inicia cuando la aplicación final indica actualizar la máquina de estados y termina cuando la acción es llevada a cabo.		
Referencias	RF.9	
Flujo Normal de los Eventos		
Acción del actor		**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación final indica actualizar la máquina de estados.		2- El sistema actualiza la máquina de estados correspondiente.
Precondiciones	-	
Postcondiciones	Se actualiza la máquina de estados.	
Flujo Alternativo		

Tabla 18. Descripción del caso de uso Actualizar Máquina de Estados.

Caso de uso	
CU-7	Enviar Mensaje
Propósito	Depositar un mensaje en el buzón dirigido a uno o más agentes.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica enviar un mensaje a un agente y termina cuando el mensaje es recibido por el otro agente.	
Referencias	RF.10
Flujo Normal de los Eventos	

Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación final indica enviar un mensaje a un agente.	2- El sistema envía el mensaje al agente destino.
Precondiciones	-
Postcondiciones	Se envía un mensaje a un agente.
Flujo Alternativo	
	2- Si el mensaje tiene retardo se deja en el buzón.

Tabla 19. Enviar Mensaje.

Caso de uso	
CU-8	Enviar Mensaje Retardados.
Propósito	Enviar los mensajes con tiempo de retardo a uno o más agentes.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica enviar mensajes existentes en el buzón que tengan tiempo de retardo.	
Referencias	RF.11
Flujo Normal de los Eventos	
Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación final indica liberar los mensajes del buzón con el retardo vencido.	2- El sistema libera los mensajes.
Precondiciones	-
Postcondiciones	Se liberan los mensajes con retardo vencido.
Flujo Alternativo	

Tabla 20. Descripción del caso de uso Enviar Mensaje de Retardo.

Caso de uso

CU-9	Gestionar Entidad
Propósito	Adiciona o eliminar entidades.
Actores: Aplicación Final (inicia).	
Resumen: El caso de uso se inicia cuando la aplicación final indica agregar o eliminar una entidad en el administrador de entidades y culmina cuando esta acción ha sido llevada a cabo.	
Referencias	RF.12, RF.13
Flujo Normal de los Eventos	
Sección "Adicionar Entidad"	
Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación final indica agregar una nueva entidad al administrador de entidades.	2- Si el id de la entidad no existe se adiciona la entidad al administrador de entidades.
Sección "Eliminar Entidad"	
Acción del actor	**Respuesta del sistema**
1- El caso de uso se inicia cuando la aplicación final indica eliminar una entidad del administrador de entidades.	2- Si el id de la entidad existe se elimina la entidad del administrador de entidades.
Precondiciones	-
Postcondiciones	Se realiza la operación especificada.
Flujos Alternativos	
Sección "Adicionar Entidad"	
	2- Si el id de la entidad existe se lanza una excepción.
Sección "Eliminar Entidad"	
	2- Si el id de la entidad no existe se lanza una excepción.

Tabla 21. Descripción del caso de uso Gestionar Entidad.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

Introducción

En este capítulo se modela el sistema y se define su forma para que soporte todos los requisitos, o sea, se presenta el modelo de diseño del sistema.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto visual en el sistema a considerar. Además el modelo de diseño sirve de abstracción a la implementación del sistema y es, de ese modo, utilizado como entrada fundamental de las actividades de implementación.

En la implementación se parte del resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, entre otros. La mayor parte de la arquitectura es capturada durante el diseño, siendo el propósito principal de la implementación el desarrollo de la arquitectura y el sistema como un todo en un lenguaje de programación específico.

3.1. Diagramas de Secuencia

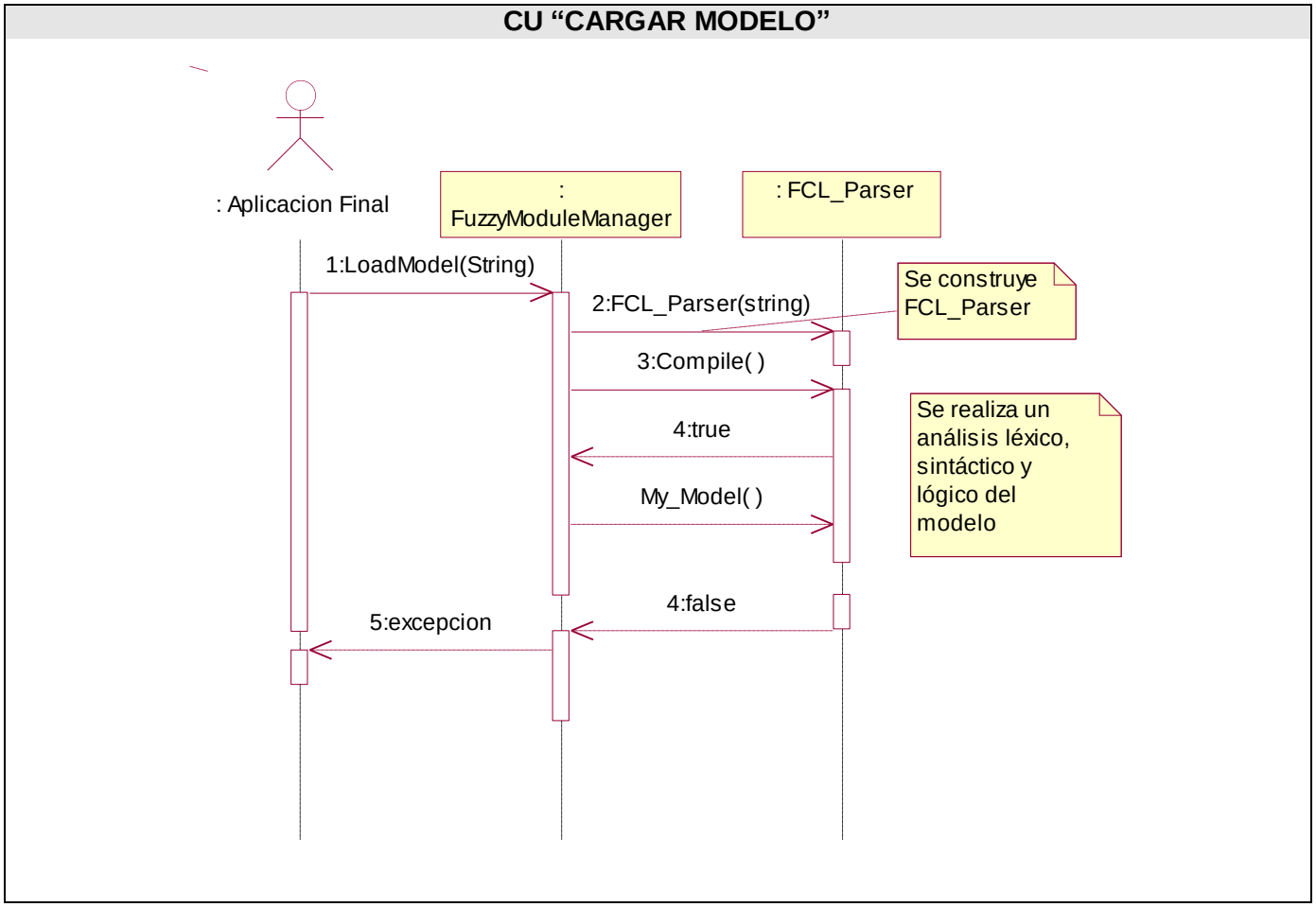

Figura 15. Diagrama de Secuencia CU Cargar Modelo

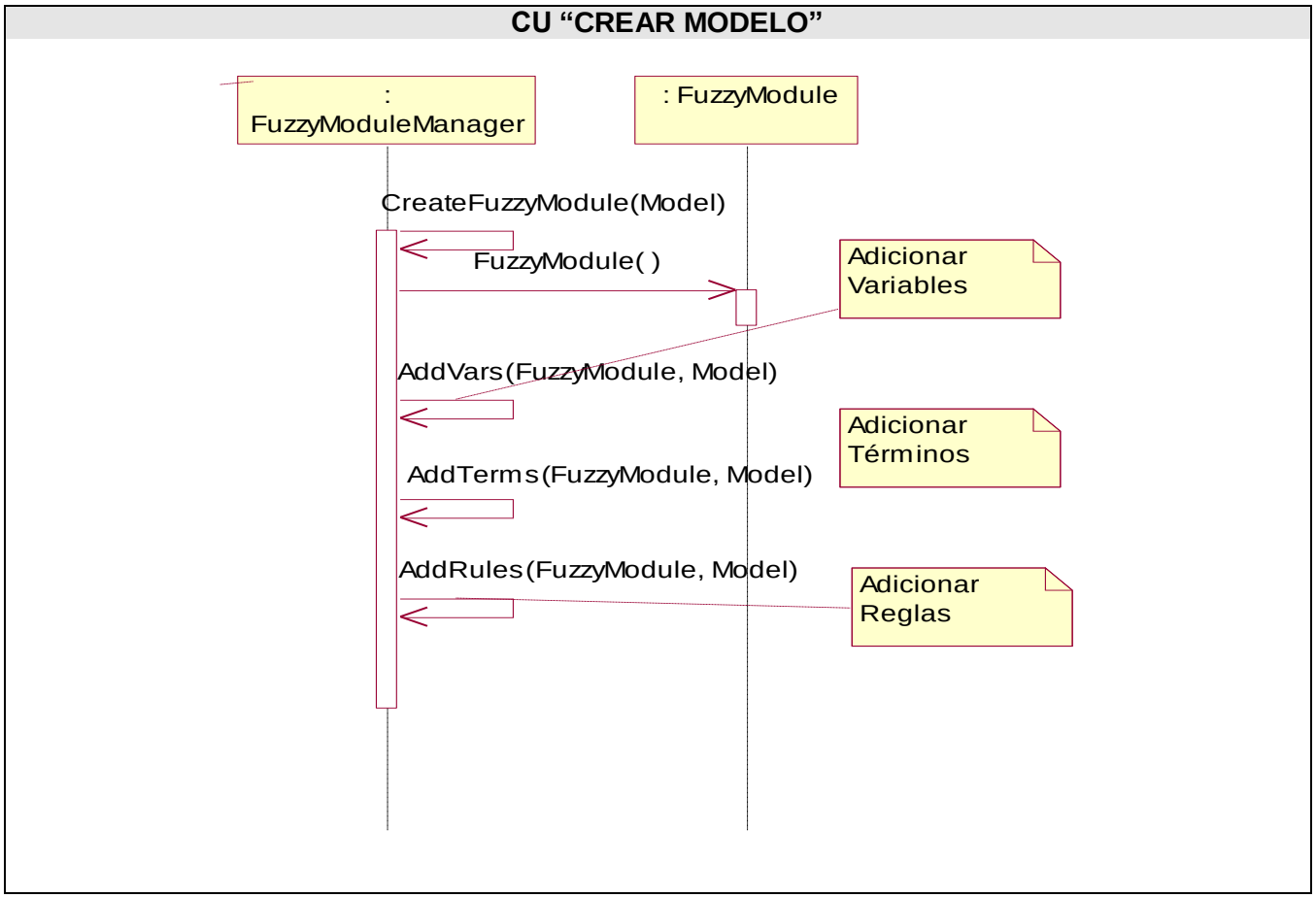

Figura 16. Diagrama de Secuencia CU Crear Modelo.

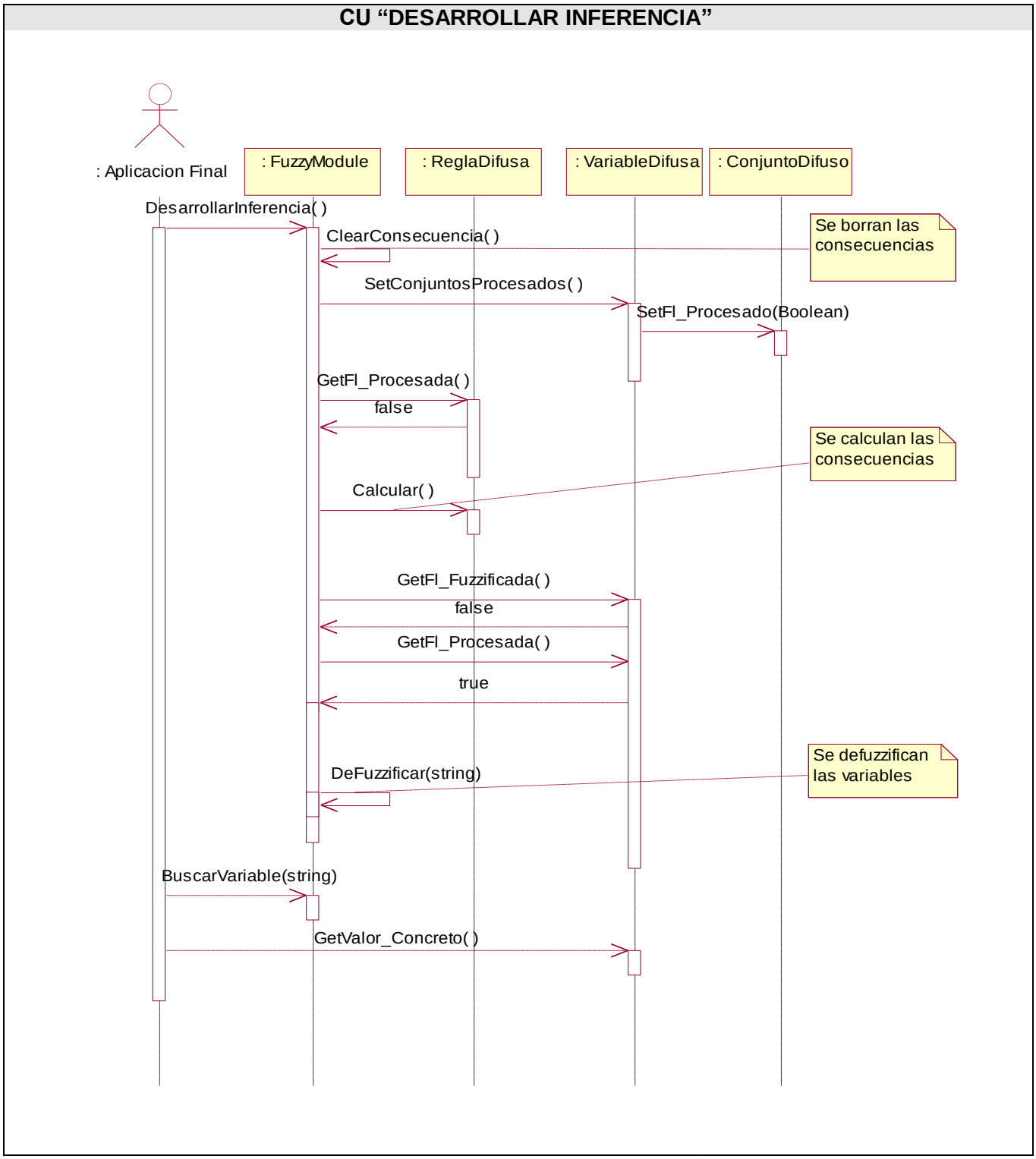

Figura 17. Diagrama de Secuencia CU Desarrollar Inferencia

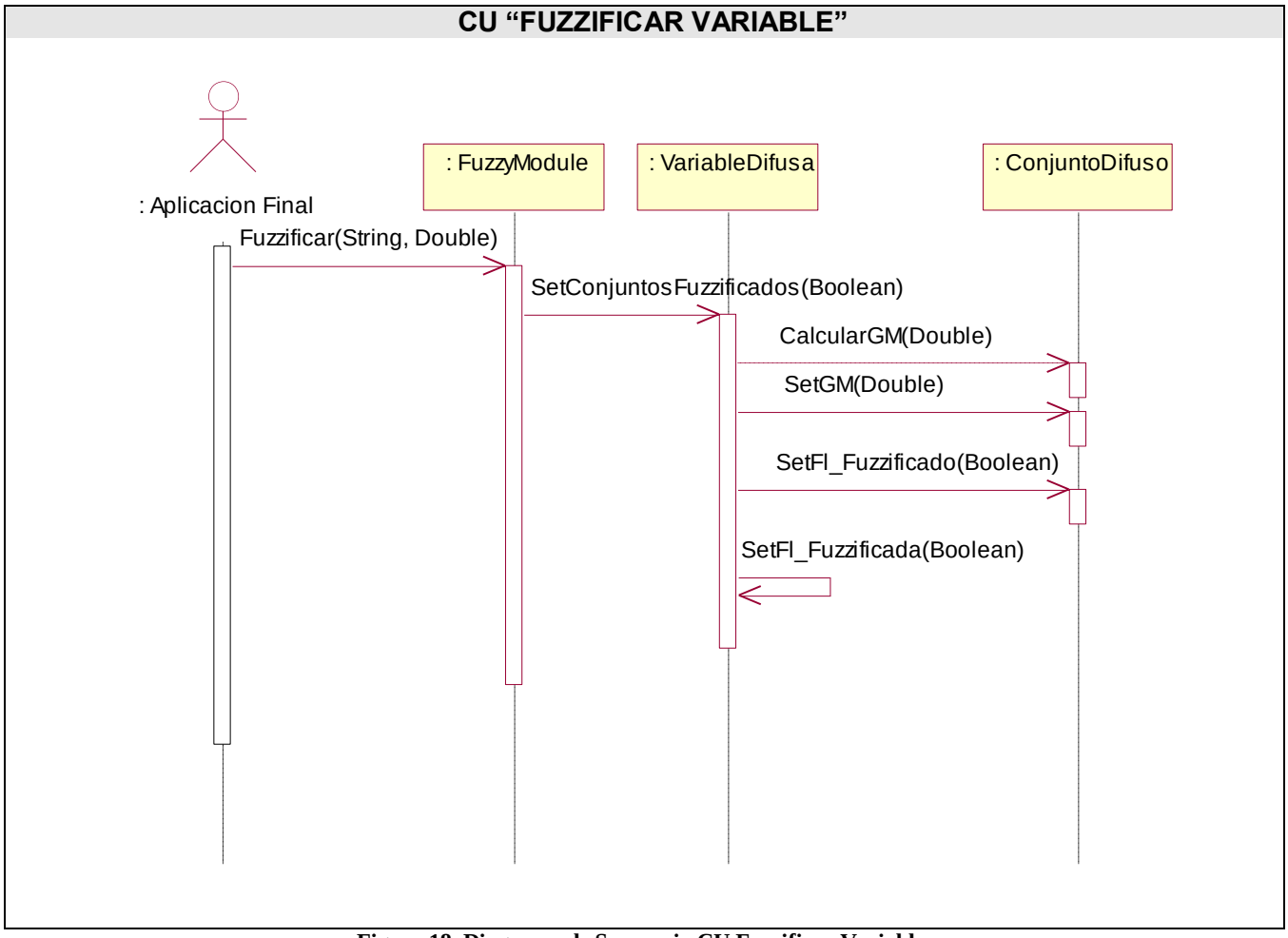

Figura 18. Diagrama de Secuencia CU Fuzzificar Variable.

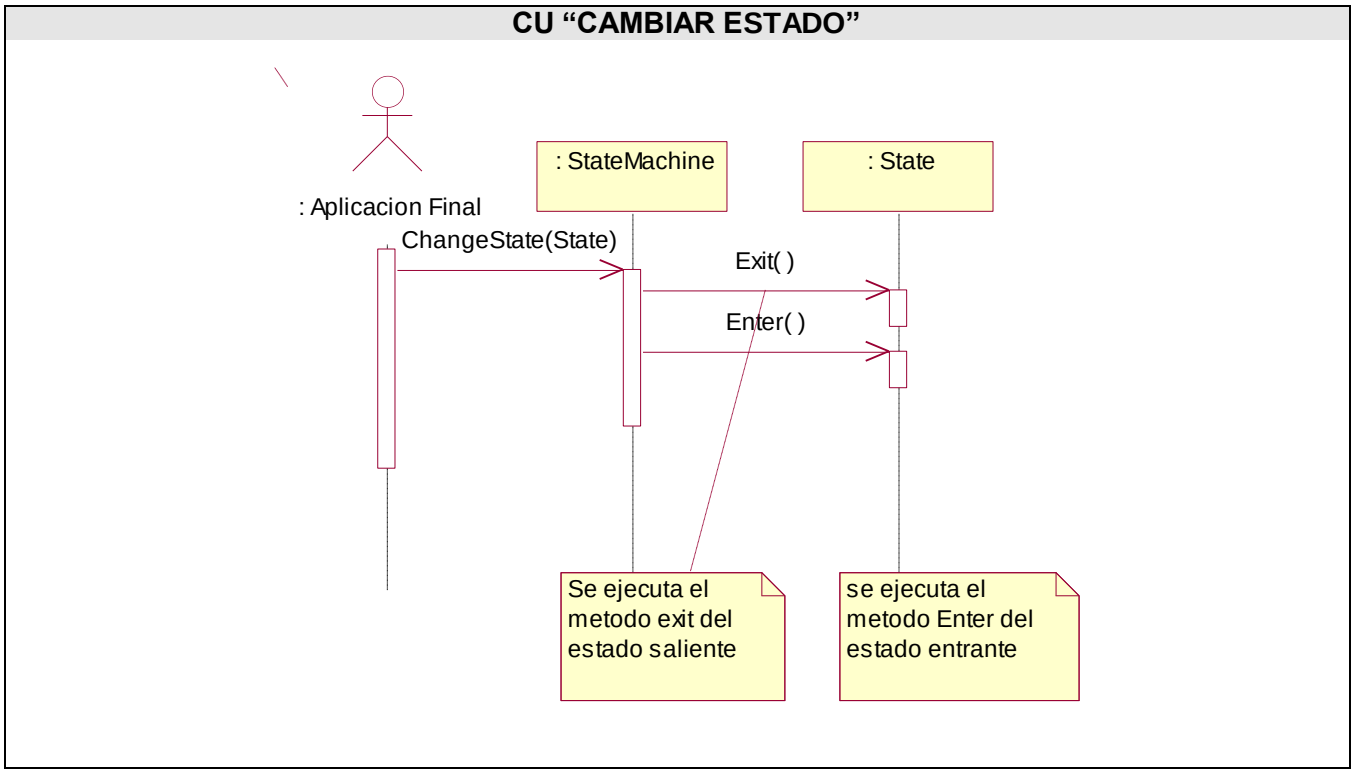

Figura 19. Diagrama de Secuencia CU Cambiar Estado

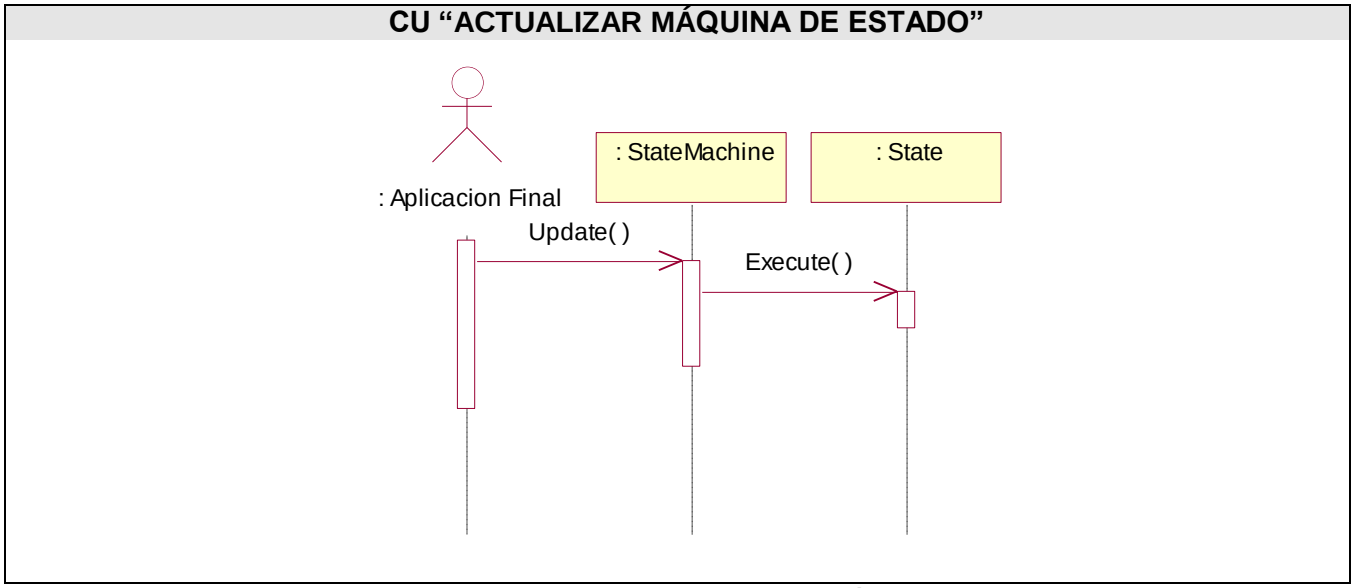

Figura 20. Diagrama de Secuencia CU Actualizar Máquina de Estados

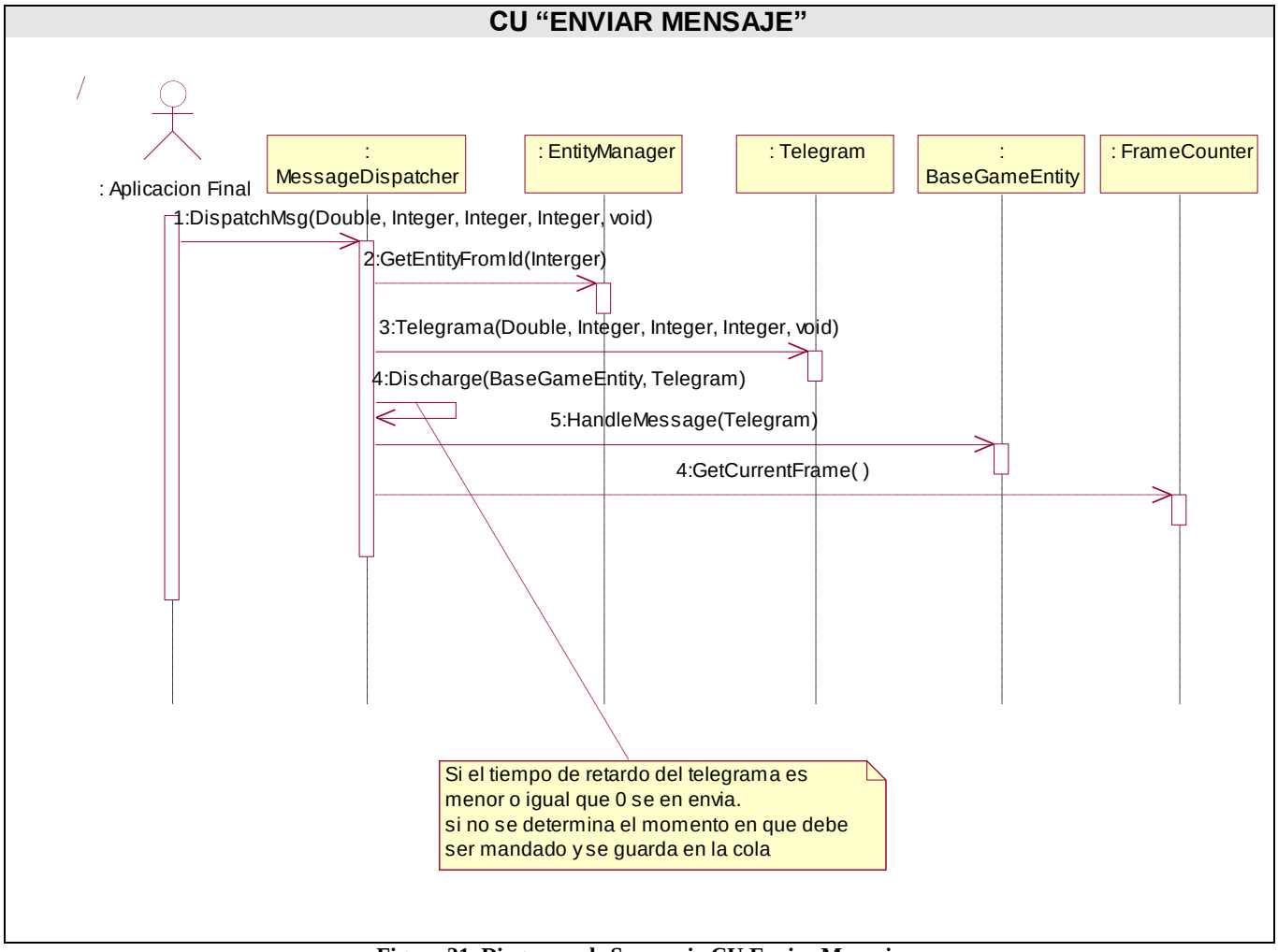

Figura 21. Diagrama de Secuencia CU Enviar Mensaje

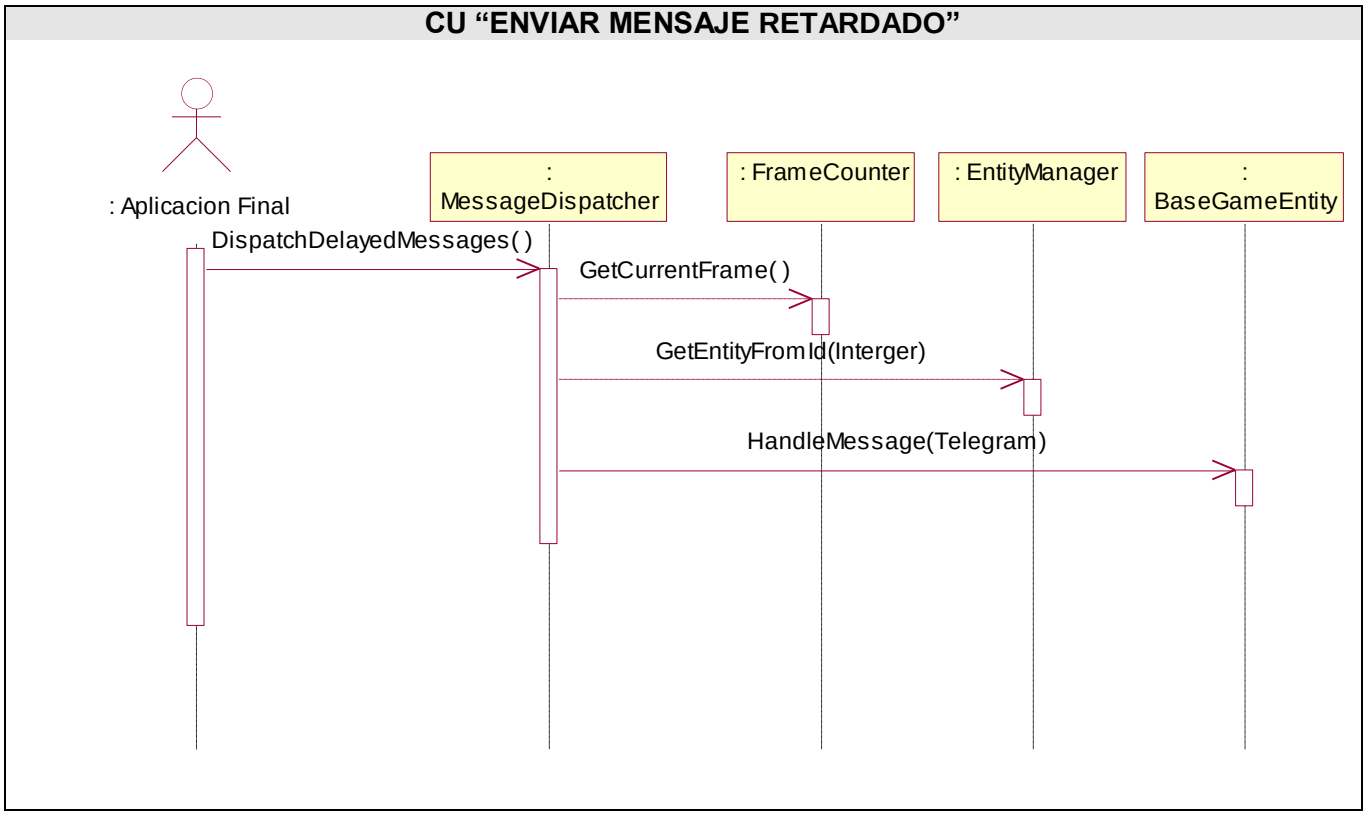

Figura 22. Diagrama de Secuencia CU Enviar Mensaje Retardado.

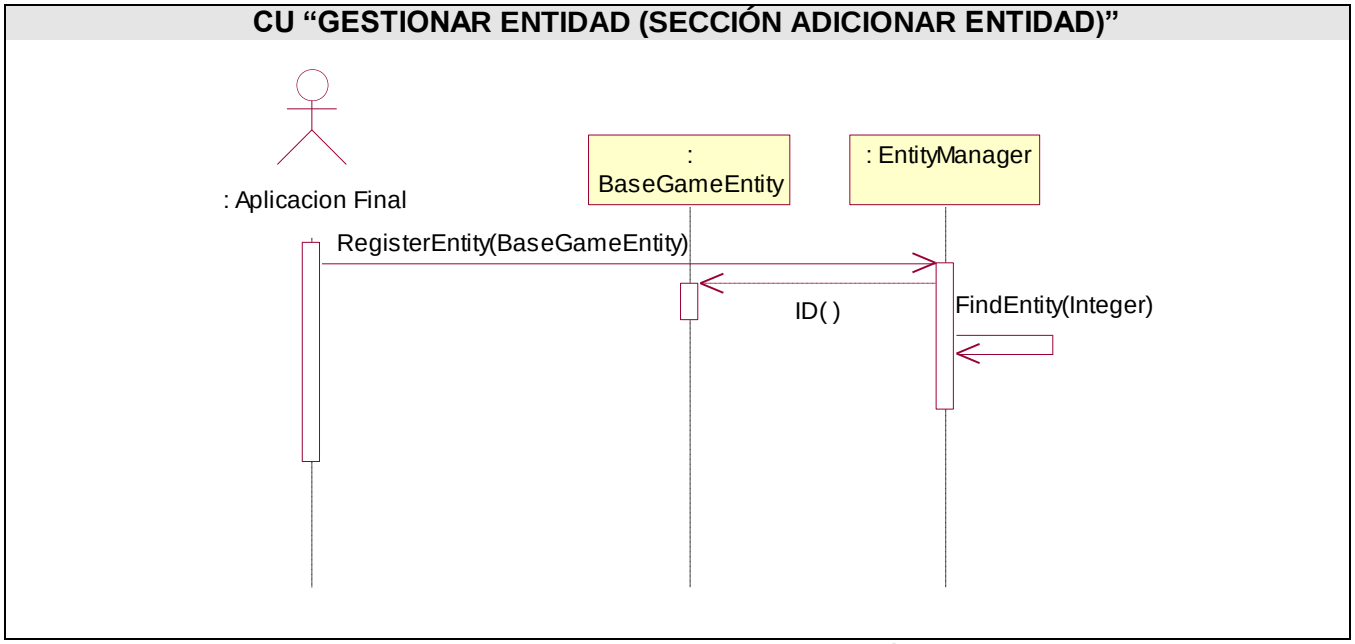

Figura 23. Diagrama de Secuencia CU Gestionar Entidad (Sección Adicionar Entidad).

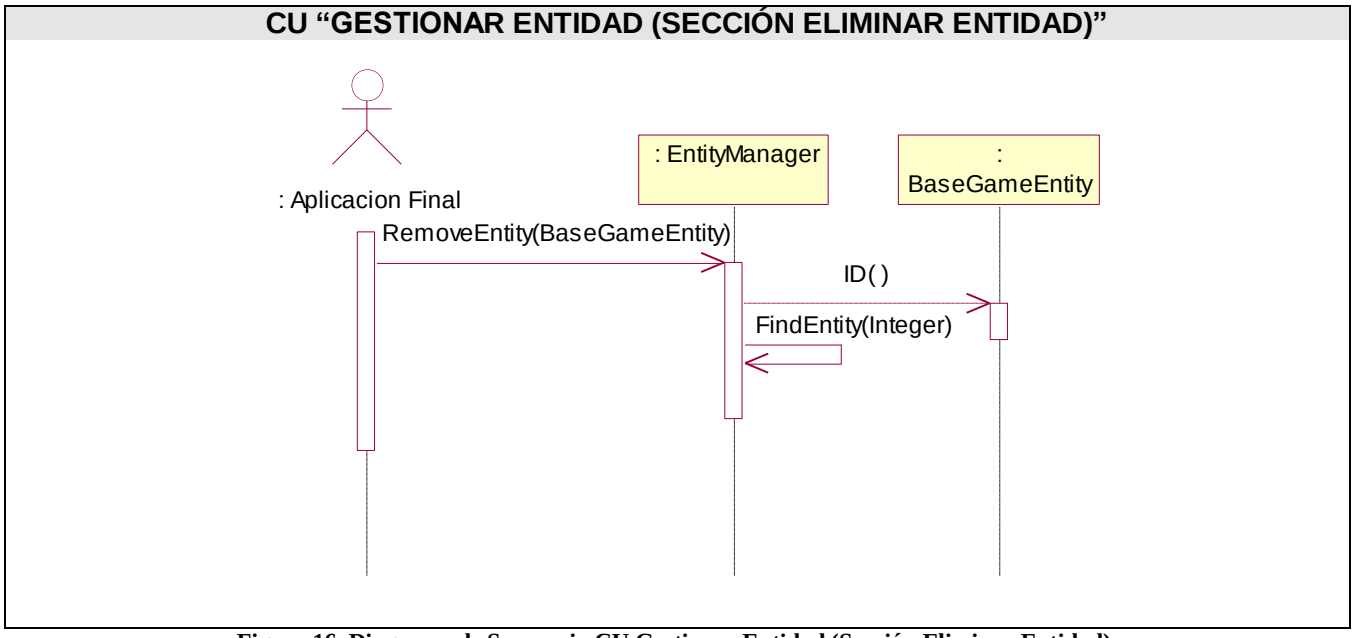

Figura 16. Diagrama de Secuencia CU Gestionar Entidad (Sección Eliminar Entidad).

3.2. Diagrama de Clases del Diseño

3.2.1. Diagrama de Clases de Diseño por paquetes

Como se hace referencia en la Propuesta del Sistema(ver epígrafe 2.1.4), el mismo estará compuesto por tres módulos principales, un módulo de lógica difusa que utiliza un módulo para interpretar un fichero de configuración con un modelo difuso basado en la especificación FCL, y un módulo que encapsula las máquinas de estados junto a un sistema de mensajes. Véase que el módulo lógica difusa depende directamente de la interpretación que haga el Parser del fichero de configuración.

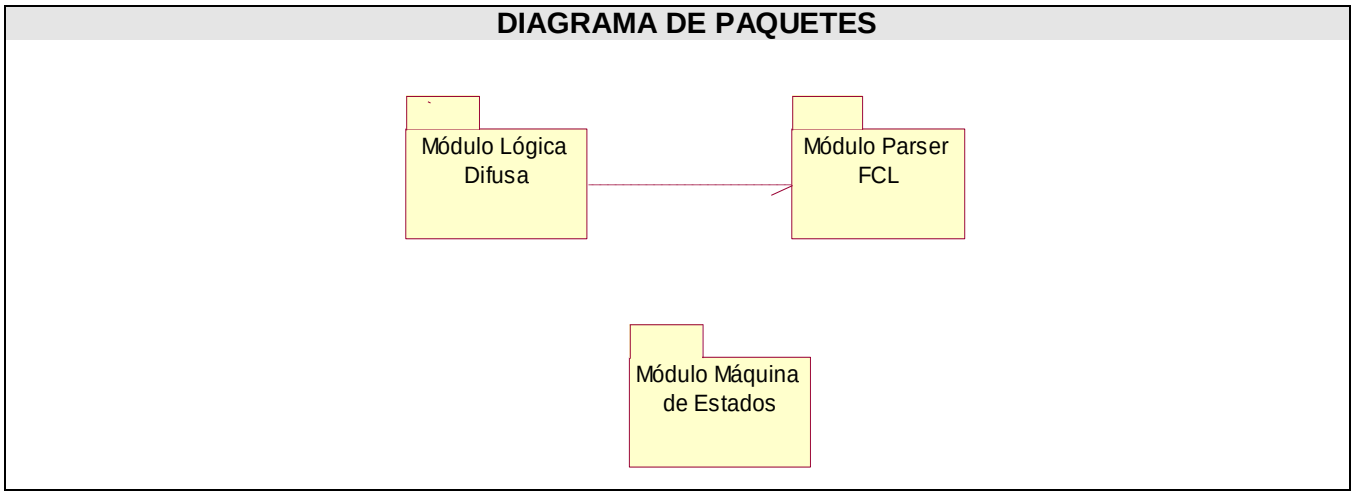

Figura 31. Diagrama de Paquetes de Clases.

3.2.2. Paquete "Módulo Lógica Difusa"

Este módulo cuenta con la clase FuzzyModuleManger que funciona como controladora e interfaz a la aplicación final; además de ser contenedora de todos los modelos cargados y creados.

La clase FuzzyModule por su parte posee todas las responsabilidades del sistema difuso específico y es contenedora de las variables y reglas del sistema.

Véase que se han implementado tres operadores: disyunción (AND), conjunción (OR), y negación (NOT), así como las funciones de contracción (VERY) y dilatación (FAERLY); ya todas estas clases se le ha definido una interfaz única que es TerminoDifuso. Además que se han implementado tres tipos de conjuntos que dependientemente de su especificación en el archivo de configuración pueden adoptar diferentes funciones de pertenencia que los representen.

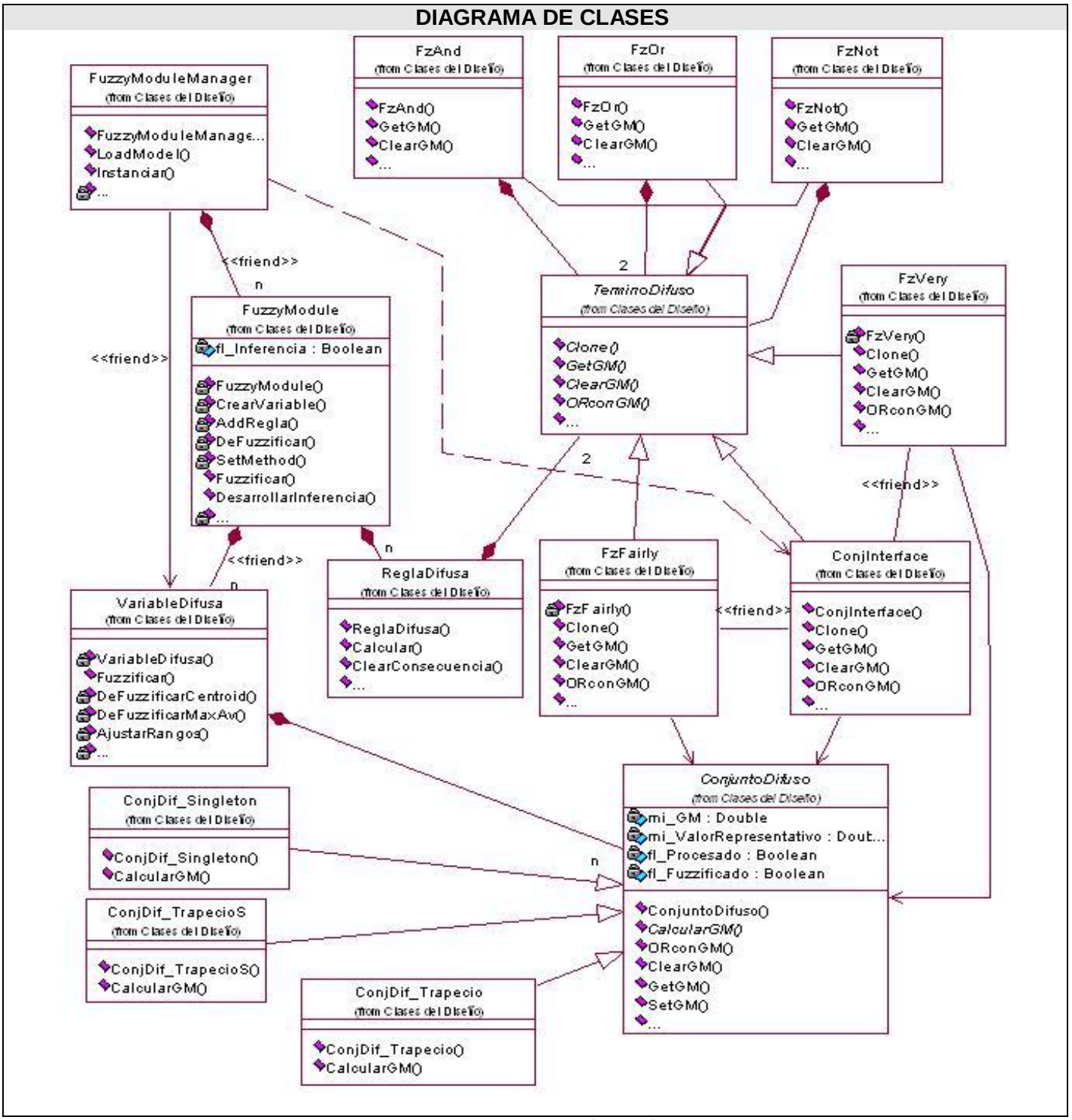

Figura 32. Diagrama de Clases Módulo Lógica Difusa

3.2.3. Paquete “Módulo Parser”

El módulo Parser tiene la responsabilidad de cargar el archivo de configuración y crear un nuevo modelo; así como generar un reporte con los errores cometidos durante la escritura del fichero de configuración.

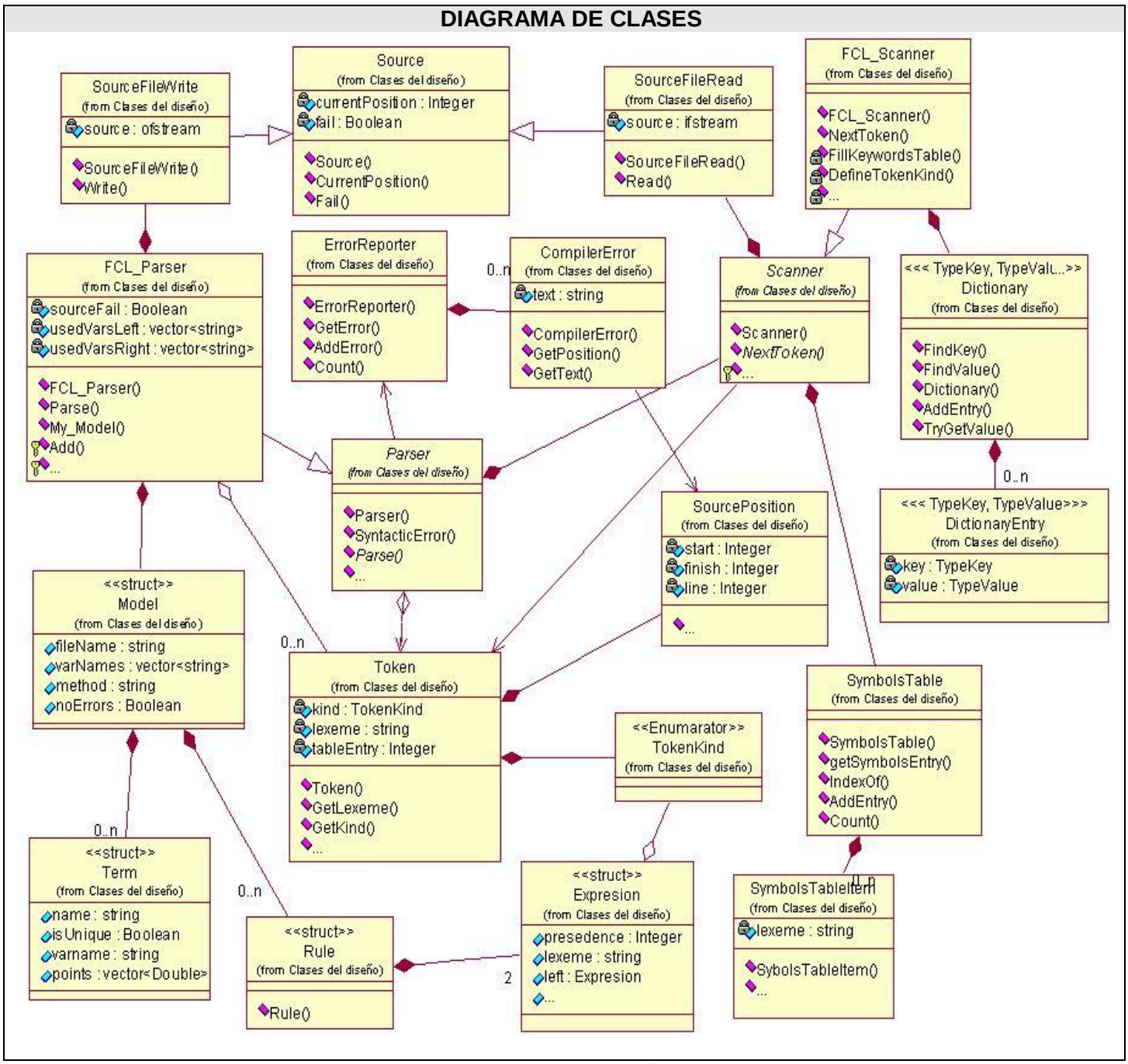

Figura 33. Diagrama de Clases Módulo Parser.

3.2.4. Paquete “Módulo Máquina de Estados”

El módulo de máquina de estados define como responsabilidades principales: La actualización y control de la máquina de estados, el envío de mensajes y la gestión de las entidades.

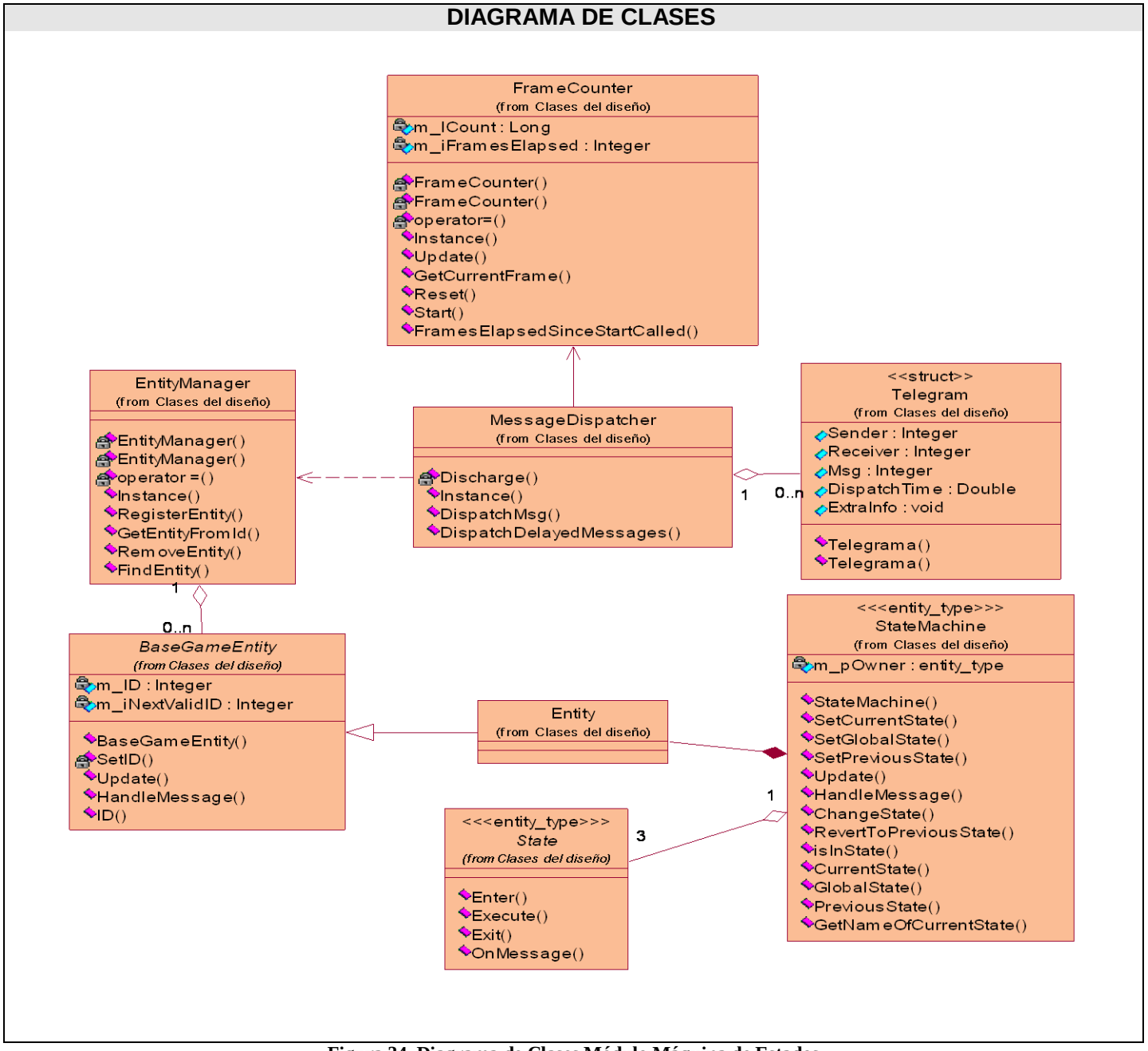

Figura 34. Diagrama de Clases Módulo Máquina de Estados

3.2.5. Relación entre las clases de los paquetes.

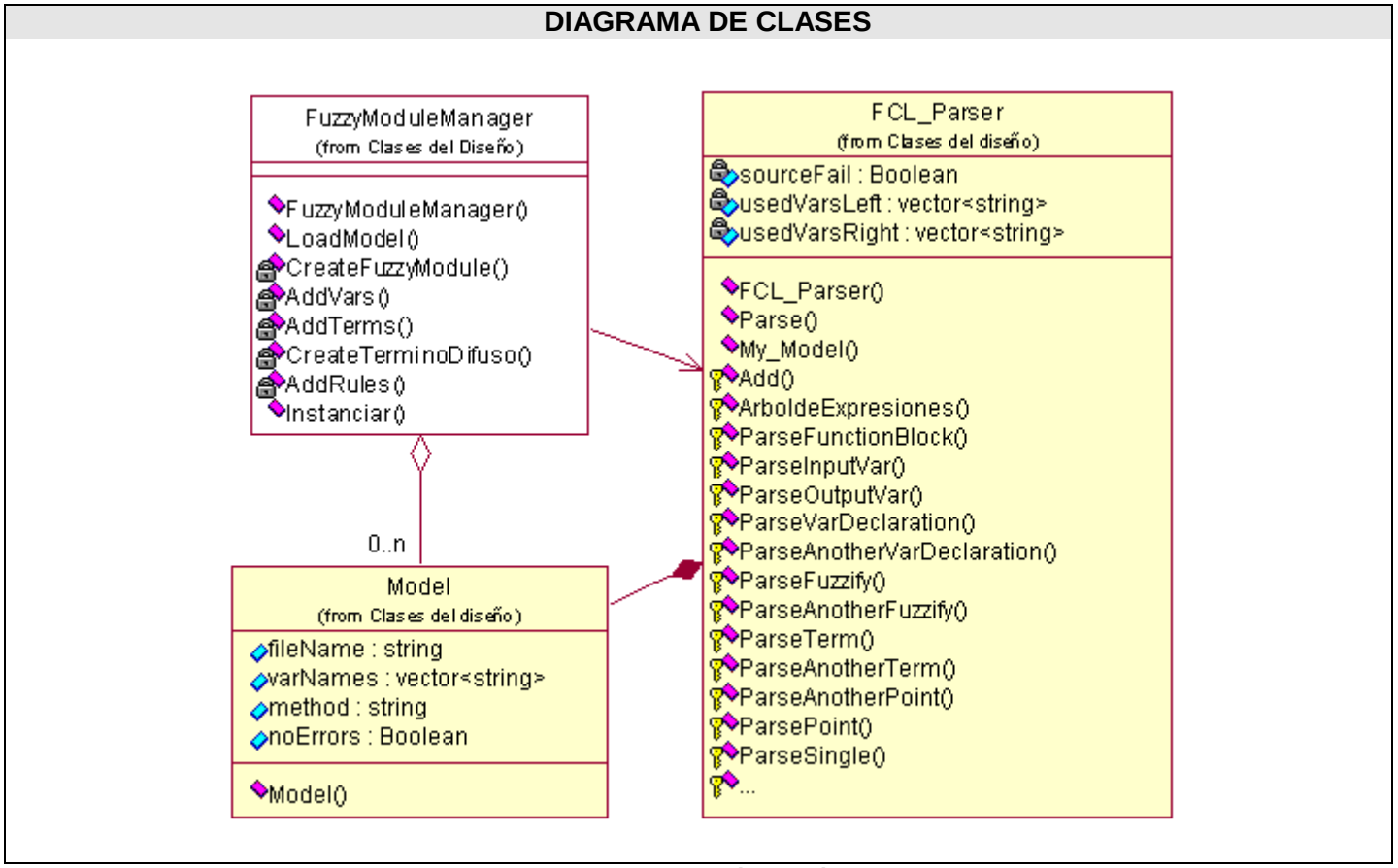

Figura 35. Diagrama de Clases Módulo Máquina de Estados

3.3. Descripción de las Clases

3.3.1. Descripción Paquete "Módulo Lógica Difusa"

Nombre: FuzzyModuleManager	
Tipo de clase : Controladora	
Atributo	**Tipo**
models	vector<Model>
modules	vector<Module>
interfaces	vector<ConjInterfaces>
Para cada responsabilidad:	

Nombre:	LoadModel
Descripción:	Compila un archivo de tipo FCL, para ello se le pasa la dirección del archivo por parámetro y crea un modelo que se guarda en el vector models además manda a ejecutar el método CreateFuzzyModule.
Nombre:	CreateFuzzyModule
Descripción:	Recibe como parámetro un modelo y su función es crear un objeto de tipo FuzzyModule si el modelo recibido no contiene errores.
Nombre:	AddVars
Descripción:	Es utilizado en la función CreateFuzzyModule, es responsable de crear las variables difusas del objeto FuzzyModule que se le pasa por parámetro.
Nombre:	AddTerms
Descripción:	Es utilizado en la función CreateFuzzyModule, es responsable de crear los conjuntos de las variables difusas del objeto FuzzyModule que se le pasa por parámetro.
Nombre:	AddRules
Descripción:	Es utilizado en la función CreateFuzzyModule, es responsable de crear las reglas difusas del objeto FuzzyModule que se le pasa por parámetro.
Nombre:	CreateTerminoDifuso
Descripción:	Es utilizado en la función AddRegla, es responsable de crear tanto el antecedente como la consecuencia de cada regla del objeto FuzzyModule.
Nombre:	Instanciar
Descripción:	Esta función tiene la responsabilidad de devolver una instancia estática de esta clase.

Tabla 22. Descripción de la clase FuzzyModuleManager

Nombre: FuzzyModule	
Tipo de clase : Controladora	
Atributo	**Tipo**
fl_Inferencia	Bool
Mis_Variables	Map<string, VariableDifusa>
Mis_Reglas	Vector<ReglaDifusa>
method	MetodoDeDefuzzificacion
Para cada responsabilidad:	
Nombre:	ClearVariables
Descripción:	Hace cero el grado de membrecía (GM) de todos los conjuntos del modelo.
Nombre:	CrearVariable

Descripción:	Crea una nueva variable difusa vacía y devuelve una referencia a ella. VarName es la llave de tipo std::string con que va a ser identificada la variable en la estructura map.
Nombre:	AddRegla
Descripción:	Adiciona una regla al módulo y recibe como parámetros los términos difusos que serán su antecedente y su consecuencia.
Nombre:	Fuzzificar
Descripción:	Este método llama al método Fuzzificar de la variable con el nombre que se pasa por parámetro.
Nombre:	BuscarVariable
Descripción:	Recibe el nombre de la variable en la estructura map y devuelve la variable correspondiente.
Nombre:	SetMethod
Descripción:	Cambia el método que será utilizado en la defuzzificación.
Nombre:	DesarrollarInferencia
Descripción:	Busca todas las reglas que contengan en su antecedente solo conjuntos difusos que pertenezcan a variables que ya hayan sido fuzzificadas y calcula el DOM de los términos difusos referido en su consecuencia. Después defuzzifica toda las variable que no hayan sido fuzzificadas y que no contengan conjuntos que no han podido ser procesados en alguna de las reglas, y acto seguido las fuzzifica para los valores que dio como resultado su defuzzificación y vuelve a ocurrir todo el proceso hasta que no quede ninguna variables sin fuzzificar o si en la iteración anterior no se produjo ninguna defuzzificación.
Nombre:	DeFuzzificar
Descripción:	Defuzzifica una variable recibe como parámetro el nombre de la variable a defuzzificar.

Tabla 23. Descripción de la clase FuzzyModule

Nombre: VariableDifusa	
Tipo de clase : Entidad	
Atributo	**Tipo**
Mis_Miembros	map<string, ConjuntoDifuso>
mi_LimInf	Double
mi_LimSup	Double
fl_Fuzzificada	Bool
fl_Procesada	Bool
valor_Concreto	Double
Para cada responsabilidad:	

Nombre:	AjustarRangos
Descripción:	Esta función tiene la responsabilidad de ejecutarse cada vez que se agrega un conjunto a la variable, recibe por parámetros los valores máximo y mínimo del conjunto para actualizar los valores máximo y mínimo del rango de la variable.
Nombre:	SetFl_Fuzzificada
Descripción:	Cambia el valor de fl_Fuzzificada, este valor se cambia a true cada vez que la variable ha sido fuzzificada y a false cada vez que se termina de ejecutar el método DesarrollarInferencia.
Nombre:	SetFl_Procesada
Descripción:	Cambia el valor de fl_Procesada, este valor cambia a true cuando todos los conjuntos de esta variable fueron procesados con éxito y a false al principio de la ejecución del método DesarrollarInferencia
Nombre:	GetFl_Fuzzificada
Descripción:	Devuelve el valor de fl_Fuzzificada
Nombre:	GetFl_Procesada
Descripción:	Devuelve el valor de fl_Procesada
Nombre:	SetConjuntosProcesados
Descripción:	Manda a ejecutar el método SetFl_Procesado en cada uno de sus conjuntos y les pasa de el valor true. Este método es llamado en el comienzo del método DesarrollarInferencia
Nombre:	SetConjuntosFuzzificados
Descripción:	Esta función tiene la responsabilidad de cambiar la bandera fl+fuzzificado de todos los conjuntos difusos que contiene esta variable.
Nombre:	GetValor_Concreto
Descripción:	Devuelve el valor del valor_Concreto
Nombre:	SetValor_Concreto
Descripción:	Cambia el valor del valor_Concreto
Nombre:	AddTrapecio
Descripción:	Adiciona un conjunto que posee una función de pertenencia de tipo Trapecio a la estructura map llamada mis_Miembros.
Nombre:	AddTrapecioS
Descripción:	Adiciona un conjunto que posee una función de pertenencia de tipo TrapecioS a la estructura map llamada mis_Miembros.
Nombre:	AddSingleton

Descripción:	Adiciona un conjunto que posee una función de pertenencia de tipo Singleton a la estructura map llamada mis_Miembros.
Nombre:	Fuzzificar
Descripción:	Fuzzifica un valor calculando el GM en cada uno de sus conjuntos.
Nombre:	DeFuzzificarMaxAv
Descripción:	Defuzzifica la variable usando el método Centro de Area.
Nombre:	DeFuzzificarCentroid
Descripción:	Defuzzifica la variable usando el método del Centroide o Centro de Gravedad.
Nombre:	ClearConjuntos
Descripción:	Esta función tiene la responsabilidad de hacer cero el grado de pertenencia de todos los conjuntos de esta variable.

Tabla 24. Descripción de la clase VariableDifusa

Nombre: ReglaDifusa	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_Antecedente	TerminoDifuso
mi_Consecuencia	TerminoDifuso
fl_Procesada	Bool
Para cada responsabilidad:	
Nombre:	SetFl_Procesada
Descripción:	Cambia el valor de fl_Procesada, este valor se cambia a false al final de la ejecución del método DesarrollarInferencia y a true cada vez que es procesada correctamente una regla.
Nombre:	GetFl_Procesada
Descripción:	Devuelve el valor de fl_Procesada.
Nombre:	ClearConsecuencia
Descripción:	Este método hace cero el GM de los conjuntos que son referido en la consecuencia de esta regla.
Nombre:	Calcular
Descripción:	Este método actualiza del GM de su consecuencia con el GM de su antecedente si ya todos los conjuntos que son referenciados en su antecedente pertenecen a variables fuzzificadas, sino ejecuta el método SetFl_Procesado de su consecuencia.

Tabla 25. Descripción de la clase ReglaDifusa

Nombre: TerminoDifuso	
Tipo de clase : Entidad	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene la responsabilidad de devolver una referencia al objeto clonado.
Nombre:	GetGM
Descripción:	Esta función tiene la responsabilidad de devolver el DOM del término difuso.
Nombre:	ClearGM
Descripción:	Esta función tiene la responsabilidad de hacer el DOM 0.
Nombre:	ORconGM
Descripción:	Esta función tiene la responsabilidad de actualizar el DOM de una consecuencia cuando esta es activada.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene la responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene la responsabilidad cambiar el valor de fl_Procesado

Tabla 26. Descripción de la clase TerminoDifuso

Nombre: ConjuntoDifuso	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_GM	Double
fl_Fuzzificado	Bool
fl_Procesado	Bool
mi_ValorRepresentativo	Double
Para cada responsabilidad:	
Nombre:	CalcularGM
Descripción:	Retorna el GM en este conjunto para un valor dado.
Nombre:	ORconGM
Descripción:	Si el conjunto pertenece a una variable que es consecuencia de una regla y este es disparado por la regla entonces este método cambia el GM (en este contexto, el GM representa un nivel de confianza) para el máximo valor entrado por parámetro.
Nombre:	GetValorRepresentativo

Descripción:	Esta función tiene la responsabilidad de devolver el valor representativo del conjunto que es el valor para el cual el conjunto tiene un mayor GM
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable fl_Fuzzificado
Nombre:	GetFl_Procesado
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable fl_Procesado
Nombre:	SetFl_Fuzzificado
Descripción:	Esta función tiene la responsabilidad de cambiar el valor de la variable fl_Fuzzificado
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene la responsabilidad de cambiar el valor de la variable fl_Procesado
Nombre:	ClearGM
Descripción:	Esta función tiene la responsabilidad de hacer cero el GM.
Nombre:	GetGM
Descripción:	Esta función tiene la responsabilidad de devolver el valor de grado de pertenencia.
Nombre:	SetGM
Descripción:	Esta función tiene la responsabilidad de cambiar el valor de grado de pertenencia.

Tabla 27. Descripción de la clase ConjuntoDifuso

Nombre: FzAnd	
Tipo de clase : Entidad	
Atributo	**Tipo**
left	TerminoDifuso
right	TerminoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia de este objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado

Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.
Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad calcular el GM correspondiente a la intersección de todos los términos difusos que contiene el vector mis_Terms.
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad hacer cero el GM de todos los términos difusos que contiene el vector mis_Terms.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad ejecutar el método ORconGM de todos los términos difusos que contiene el vector mis_Terms.

Tabla 28. Descripción de la clase FzAnd

Nombre: FzOR	
Tipo de clase : Entidad	
Atributo	**Tipo**
left	TerminoDifuso
right	TerminoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia de este objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.
Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad calcular el GM correspondiente a la unión de todos los términos difusos que contiene el vector mis_Terms
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad hacer cero el GM de todos los términos difusos que contiene el vector mis_Terms.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad ejecutar el método ORconGM de todos los términos difusos que contiene el vector mis_Terms

Tabla 29. Descripción de la clase FzOr

Nombre: FzNot	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_Termino	TerminoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia de este objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.
Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad calcular el GM correspondiente a la negación de todos los términos difusos que contiene el vector mis_Terms.
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad hacer cero el GM de todos los términos difusos que contiene el vector mis_Terms.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad ejecutar el método ORconGM de todos los términos difusos que contiene el vector mis_Terms.

Tabla 30. Descripción de la clase FzNot

Nombre: FzVery	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_Conjunto	TConjuntoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia de este objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.

Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad calcular el cuadrado del GM del conjunto difuso que esta clase representa.
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad hacer cero el GM del conjunto difuso que esta clase representa.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad ejecutar el método ORconGM del conjunto difuso que esta clase representa.

Tabla 31. Descripción de la clase FzVery

Nombre: FzFairly	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_Conjunto	TConjuntoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia a una copia del objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.
Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad de calcular la raíz cuadrada del GM del conjunto difuso que esta clase representa.
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad de hacer cero el GM del conjunto difuso que esta clase representa.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad de ejecutar el método ORconGM del conjunto difuso que esta clase representa.

Tabla 32. Descripción de la clase FzFairly

Nombre: ConjDif_Trapecio	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_PicoIzq	double
mi_PicoDer	double
mi_LeftOffset	double
mi_RightOffset	double
Para cada responsabilidad:	
Nombre:	CalcularGM
Descripción:	Esta función tiene como responsabilidad calcular el GM de un valor para la función de pertenencia que posee este conjunto difuso.

Tabla 33. Descripción de la clase ConjDif_Trapecio

Nombre: ConjDif_TrapecioS	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_PicoIzq	double
mi_PicoDer	double
mi_LeftOffset	double
mi_RightOffset	double
punto_inflex_izq	double
punto_inflex_der	double
Para cada responsabilidad:	
Nombre:	CalcularGM
Descripción:	Esta función tiene como responsabilidad calcular el GM de un valor para la función de pertenencia que posee este conjunto difuso.

Tabla 34. Descripción de la clase ConjDif_TrapecioS

Nombre: ConjDif_Singleton	
Tipo de clase : Entidad	
Atributo	**Tipo**
mi_Pico	double
Para cada responsabilidad:	
Nombre:	CalcularGM
Descripción:	Esta función tiene como responsabilidad calcular el GM de un valor para la función de pertenencia que posee este conjunto difuso.

Tabla 35. Descripción de la clase ConjDif_Singleton

Nombre: ConjInterface	
Tipo de clase :	
Atributo	**Tipo**
mi_Conjunto	ConjuntoDifuso
Para cada responsabilidad:	
Nombre:	Clone
Descripción:	Esta función tiene como responsabilidad devolver una referencia de este objeto.
Nombre:	GetFl_Fuzzificado
Descripción:	Esta función tiene como responsabilidad devolver el valor de fl_Fuzzificado.
Nombre:	SetFl_Procesado
Descripción:	Esta función tiene como responsabilidad cambiar el valor de fl_Procesado.
Nombre:	GetGM
Descripción:	Esta función tiene como responsabilidad devolver el GM del conjunto difuso que esta clase representa.
Nombre:	ClearGM
Descripción:	Esta función tiene como responsabilidad hacer cero el GM del conjunto difuso que esta clase representa.
Nombre:	ORconGM
Descripción:	Esta función tiene como responsabilidad ejecutar el método ORconGM del conjunto difuso que esta clase representa.

Tabla 36. Descripción de la clase ConjInterface

3.3.2. Descripción Paquete "Módulo Parser FCL"

Nombre: SourcePosition	
Tipo de clase : Entidad	
Atributo	**Tipo**
Start	int
Finish	int
Line	int
Para cada responsabilidad:	
Nombre:	GetStart
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable start.
Nombre:	GetFinish

Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable finish.
Nombre:	GetLine
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable line

Tabla 37. Descripción de la clase SourcePosition

Nombre: Token	
Tipo de clase : Entidad	
Atributo	**Tipo**
lexeme	string
Kind	TokenKind
tableEntry	int
sourcePosition	SourcePosition
Para cada responsabilidad:	
Nombre:	GetLexeme
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable lexeme.
Nombre:	GetKind
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable kind.
Nombre:	GetInSourcePosition
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable tableEntry.
Nombre:	GetTableEntry
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable sourcePosition.
Nombre:	TokenKindString
Descripción:	Devuelve un valor de tipo string correspondiente al tipo de token que se le pasa por parámetro.

Tabla 38. Descripción de la clase Token

Nombre: SymbolsTableItem	
Tipo de clase : Entidad	
Atributo	**Tipo**
lexeme	string
type	int
dir	int
declared	bool
initialized	bool
Para cada responsabilidad:	
Nombre:	getLexeme

Descripción:	Devuelve un valor de tipo string correspondiente al tipo de token que se le pasa por lexeme.

Tabla 39. Descripción de la clase SymbolsTableItem

Nombre: SymbolsTable	
Tipo de clase : Controladora	
Atributo	**Tipo**
items	vector< SymbolsTableItem >
Para cada responsabilidad:	
Nombre:	IndexOf
Descripción:	Esta función tiene como responsabilidad devolver la dirección que ocupa el lexema pasado por parámetro en la tabla de símbolos.
Nombre:	AddEntry
Descripción:	Esta función tiene como responsabilidad adicionar un nuevo lexema a la tabla de símbolos.
Nombre:	Count
Descripción:	Esta función tiene como responsabilidad devolver la cantidad de símbolos que existen en la tabla de símbolos.

Tabla 40. Descripción de la clase SymbolsTable

Nombre: SourceFileRead	
Tipo de clase : Entidad	
Atributo	**Tipo**
currentPosition	int
source	ifstream
fail	bool
Para cada responsabilidad:	
Nombre:	Read
Descripción:	Esta función tiene como responsabilidad leer un carácter del fichero.
Nombre:	CurrentPosition
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable currentPosition.
Nombre:	Fail
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable fail.

Tabla 41. Descripción de la clase SourceFileRead

Nombre: SourceFileWrite

Tipo de clase : Entidad	
Atributo	**Tipo**
currentPosition	int
source	ifstream
fail	bool
Para cada responsabilidad:	
Nombre:	Write
Descripción:	Esta función tiene como responsabilidad escribir una línea del fichero.
Nombre:	CurrentPosition
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable currentPosition.
Nombre:	Fail
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable fail.

Tabla 42. Descripción de la clase SourceFileWrite

Nombre: DictionaryEntry	
Tipo de clase : Entidad	
Atributo	**Tipo**
key	TypeKey
value	TypeValue
Para cada responsabilidad:	
Nombre:	GetKey
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable key.
Nombre:	GetValue
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable value.

Tabla 43. Descripción de la clase DictionaryEntry

Nombre: Dictionary	
Tipo de clase : Controladora	
Atributo	**Tipo**
valList	vector< DictionaryEntry >
Para cada responsabilidad:	
Nombre:	FindKey
Descripción:	Esta función tiene la responsabilidad de devolver la dirección del la primera entrada que tenga como valor de su llave el valor entrado por parámetro.
Nombre:	FindValue
Descripción:	Esta función tiene la responsabilidad de devolver la dirección del la primera entrada que tenga el valor entrado por parámetro.

Nombre:	AddEntry
Descripción:	Esta función tiene la responsabilidad de adicionar una nueva entrada al diccionario.
Nombre:	TryGetValue
Descripción:	Esta función tiene la responsabilidad de devolver un booleano que responde a la existencia de una entrada determinada en el diccionario.

Tabla 44. Descripción de la clase Dictionary

Nombre: Scanner	
Tipo de clase : Controladora	
Atributo	**Tipo**
source	SourceFileRead
symbolsTable	SymbolsTable
Para cada responsabilidad:	
Nombre:	IsLetter
Descripción:	Esta función tiene la responsabilidad de detectar si un carácter es una letra.
Nombre:	IsDigit
Descripción:	Esta función tiene la responsabilidad de detectar si un carácter es un dígito.
Nombre:	NextChar
Descripción:	Esta función tiene la responsabilidad orientar al objeto source que lee otro carácter.
Nombre:	NextToken
Descripción:	Esta función tiene la responsabilidad crear un nuevo token y devolverlo.

Tabla 45. Descripción de la clase Scanner

Nombre: FCL_Scanner	
Tipo de clase : Controladora	
Atributo	**Tipo**
currentLine	int
lexemeBuilder	string
currentlyScanningToken	bool
currentChar	char
keywordsTable	Dictionary< string, TokenKind>
Para cada responsabilidad:	
Nombre:	FillKeywordsTable
Descripción:	Esta función tiene la responsabilidad de definir que cadenas van a ser palabras claves para el lenguaje que reconoce este Analizador Léxico.

Nombre:	IsSeparator
Descripción:	Esta función tiene la responsabilidad de detectar si un carácter dado es un separador.
Nombre:	TakeIt
Descripción:	Esta función tiene la responsabilidad de, si se está reconociendo un token, concatenar el último carácter leído al lexema actual y lee un nuevo carácter del Stream de entrada.
Nombre:	DefineTokenKind
Descripción:	Esta función clasifica el tipo de token actual según el lexema: si está en el diccionario (keywordsTable) es una palabra reservada de lo contrario es un identificador.
Nombre:	NextToken
Descripción:	Esta función tiene la responsabilidad de devolver en cada llamada el siguiente token reconocido en el fichero o cadena de entrada.

Tabla 46. Descripción de la clase FCL_Scanner

Nombre: CompilerError	
Tipo de clase : Entidad	
Atributo	**Tipo**
text	string
position	SourcePosition
Para cada responsabilidad:	
Nombre:	GetText
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable text.
Nombre:	GetPosition
Descripción:	Esta función tiene la responsabilidad de devolver el valor de la variable position.

Tabla 47. Descripción de la clase CompilerError

Nombre: ErrorReporter	
Tipo de clase : Controladora	
Atributo	**Tipo**
errors	vector< CompilerError >
Para cada responsabilidad:	
Nombre:	AddError
Descripción:	Esta función tiene la responsabilidad de adicionar un nuevo error.
Nombre:	GetError

Descripción:	Esta función tiene la responsabilidad de devolver el error que se encuentre en una posición dada del vector errors.
Nombre:	Count
Descripción:	Esta función tiene la responsabilidad de devolver la cantidad de errores que contiene este objeto

Tabla 48. Descripción de la clase ErrorReporter

Nombre: Parser	
Tipo de clase : Controladora	
Atributo	**Tipo**
scanner	Scanner
errorReporter	ErrorReporter
currentToken	Token
Para cada responsabilidad:	
Nombre:	SyntacticError
Descripción:	Esta función tiene la responsabilidad de adicionar un nuevo error al reporte de errores.
Nombre:	Accept
Descripción:	Esta función tiene la responsabilidad de pedir al scanner un nuevo token si el token actual es del tipo que se especifica por parámetro.
Nombre:	AcceptIt
Descripción:	Esta función tiene la responsabilidad de pedir al scanner un nuevo token.

Tabla 49. Descripción de la clase Parser

Nombre: FCL_Parser	
Tipo de clase : Controladora	
Atributo	**Tipo**
tokens	vector< Token >
my_model	Model
reportFile	SourceFileWrite
sourceFail	bool
usedVarsLeft	vector<string>
usedVarsRight	vector<string>
Para cada responsabilidad:	
Nombre:	My_Model
Descripción:	Esta funcion tiene la responsabilidad de devolver el valor de my_model

Nombre:	Add
Descripción:	Adiciona un token al vector tokens que será usado para crear un árbol de expresiones.
Nombre:	ArboldeExpresiones
Descripción:	Crea un árbol de expresiones con los tokens del vector tokens. Este árbol de expresiones puede referirse a las expresiones del antecedente o de la consecuencia de una regla.
Nombre:	Parse
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de todo el modelo.
Nombre:	ParseFunctionBlock
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque de funciones.
Nombre:	ParseInputVar
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque de variables de entrada.
Nombre:	ParseOutputVar
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque de variables de salida
Nombre:	ParseVarDeclaration
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de la declaración de una variable y adicionar la variable al vector varNames
Nombre:	ParseAnotherVarDeclaration
Descripción:	Esta función tiene la responsabilidad de verificar si existe otra declaración de variable a continuación de la variable antes declarada o si se cierra el bloque.
Nombre:	ParseFuzzify
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque Fuzzify.
Nombre:	ParseAnotherFuzzify
Descripción:	Esta función tiene la responsabilidad de verificar si existe otro bloque Fuzzify a continuación del bloque antes declarado.
Nombre:	ParseTerm
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de la declaración de un término y adicionar el mismo al vector terms.
Nombre:	ParseAnotherTerm
Descripción:	Esta función tiene la responsabilidad de verificar si existe otro término después

	del anteriormente declarado o si se cierra el bloque Fuzzify
Nombre:	ParseAnotherPoint
Descripción:	Esta función tiene la responsabilidad de verificar si existe otro punto después del anteriormente declarado.
Nombre:	ParsePoint
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis declaración de un punto.
Nombre:	ParseRuleBlock
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque RuleBlock.
Nombre:	ParseAccum
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de declaración del método de acumulación.
Nombre:	ParseRule
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de declaración de una regla.
Nombre:	ParseAnotherRule
Descripción:	Esta función tiene la responsabilidad de verificar si existe otra regla después de la anteriormente declarada.
Nombre:	ParseRuleExpresion
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una expresión dentro de una regla esta expresión debe ser tanto el antecedente como la consecuencia.
Nombre:	ParseOr
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una operación de unión.
Nombre:	ParseAnotherOr
Descripción:	Esta función tiene la responsabilidad de verificar si existe otra operación de unión a continuación de la anterior.
Nombre:	ParseAnd
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una operación de intersección.
Nombre:	ParseAnotherAnd
Descripción:	Esta función tiene la responsabilidad de verificar si existe otra operación de intersección a continuación de la anterior.

Nombre:	ParseNot
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una operación de negación
Nombre:	ParseHedges
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una aproximación.
Nombre:	ParseIs
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis de una especificación de tipo IS.
Nombre:	ParseDeFuzzify
Descripción:	Esta función tiene la responsabilidad de verificar la sintaxis del bloque DeFuzzify.
Nombre:	AnalizeVars
Descripción:	Esta función tiene la responsabilidad de determinar si no hay errores lógicos en la declaración de las variables.
Nombre:	AnalizeTerms
Descripción:	Esta función tiene la responsabilidad de determinar si no hay errores lógicos en la declaración de los términos.
Nombre:	AnalizeRules
Descripción:	Esta función tiene la responsabilidad de determinar si no hay errores lógicos en la declaración de las reglas.
Nombre:	AnalizeRule
Descripción:	Esta función tiene la responsabilidad de determinar si no hay errores lógicos en la declaración de una regla en específico.
Nombre:	AnalizeExpresion
Descripción:	Esta función tiene la responsabilidad de determinar si no hay errores lógicos en la declaración de una expresión.
Nombre:	FindTerm
Descripción:	Esta función tiene la responsabilidad de determinar si existe un término que tenga un nombre dado y pertenezca a una variable dada.
Nombre:	LogicError
Descripción:	Esta función tiene la responsabilidad adicionar un error lógico al reporte de errores.
Nombre:	Compile
Descripción:	Esta función tiene la responsabilidad de iniciar el proceso de compilación de todo

	el modelo.

Tabla 50. Descripción de la clase FCL_Parser

Nombre: Term	
Tipo de clase : Entidad	
Atributo	**Tipo**
isUnique	bool
name	string
varname	string
points	vector<double>
Para cada responsabilidad:	
Nombre:	operator ==
Descripción:	Esta función tiene la responsabilidad de determinar si dos términos tienen el mismo nombre y pertenecen a la misma variable.
Nombre:	AnalizePoints
Descripción:	Esta función tiene la responsabilidad de comprobar la correcta declaración de los puntos que definen este término difuso.

Tabla 51. Descripción de la clase Term

Nombre: Expresion	
Tipo de clase : Entidad	
Atributo	**Tipo**
presedence	int
tk	TokenKind
lexeme	string
left	Expresion
right	Expresion
Para cada responsabilidad:	
Nombre:	operator =
Descripción:	Esta función tiene la responsabilidad de igualar una expresión a otra.
Nombre:	AnalizePoints
Descripción:	Esta función tiene la responsabilidad de comprobar la correcta declaración de los puntos que definen este término difuso.
Nombre:	AddExpresion
Descripción:	Esta función tiene la responsabilidad de agregar una expresión al árbol de expresiones que constituye esta instancia.

Tabla 52. Descripción de la clase Expresion

Nombre: Rule	
Tipo de clase : Entidad	
Atributo	**Tipo**
left	Expresion
right	Expresion

Tabla 53. Descripción de la clase Rule

Nombre: Model	
Tipo de clase : Entidad	
Atributo	**Tipo**
fileName	string
varNames	vector<string>
terms	vector<Term>
rules	vector<Rule>
method	string
noErrors	bool

Tabla 54. Descripción de la clase Model

3.3.3. Descripción Paquete "Módulo Máquina de Estados"

Nombre: BaseGameEntity	
Tipo de clase : Entidad	
Atributo	**Tipo**
m_ID	
m_iNextValidID	
Para cada responsabilidad:	
Nombre:	SetID
Descripción:	Esta función es llamada desde el constructor y tiene la responsabilidad de determinar si el id de esta entidad es mayor o igual que el próximo id válido.
Nombre:	Update
Descripción:	Esta función tiene la responsabilidad de actualizar el estado de la entidad en el mundo del juego.
Nombre:	HandleMessage
Descripción:	Esta función tiene la responsabilidad de manejar los mensajes que son notificados a la entidad.

Nombre:	ID
Descripción:	Esta función tiene la responsabilidad de devolver el id de la entidad

Tabla 55. Descripción de la clase BaseGameEntity

Nombre: EntityManager	
Tipo de clase : Controladora	
Atributo	Tipo
m_EntityMap	map<int, BaseGameEntity>
Para cada responsabilidad:	
Nombre:	Instance
Descripción:	Esta función tiene la responsabilidad de devolver una instancia estática de esta clase.
Nombre:	RegisterEntity
Descripción:	Esta función tiene la responsabilidad de adicionar una entidad a la estructura m_EntityMap.
Nombre:	GetEntityFromID
Descripción:	Esta función tiene la responsabilidad de buscar y devolver una instancia a la entidad que tiene como id el pasado por parámetro.
Nombre:	RemoveEntity
Descripción:	Esta función tiene la responsabilidad de eliminar una entidad dada.
Nombre:	FindEntity
Descripción:	Esta función tiene la responsabilidad de buscar una instancia a la entidad que tiene como id el pasado por parámetro. Devuelve true si lo encontró y false en caso contrario.

Tabla 56. Descripción de la clase EntityManager

Nombre: State<entity_type>	
Tipo de clase : Entidad	
Para cada responsabilidad:	
Nombre:	Enter
Descripción:	Esta función tiene la responsabilidad de ejecutar las acciones correspondientes a la entrada en este estado.
Nombre:	Execute
Descripción:	Esta función tiene la responsabilidad de ejecutar las acciones correspondientes a la ejecución de este estado.

Nombre:	Exit
Descripción:	Esta función tiene la responsabilidad de ejecutar las acciones correspondientes a la salida de este estado.
Nombre:	OnMessage
Descripción:	Esta función tiene la responsabilidad de ejecutar las acciones correspondientes al recibo de un mensaje dado en este estado.

Tabla 57. Descripción de la clase State

Nombre: StateMachine<entity_type>	
Tipo de clase : Controladora	
Atributo	**Tipo**
m_pOwner	entity_type
m_pCurrentState	State<entity_type>
m_pPreviousState	State<entity_type>
m_pGlobalState	State<entity_type>
Para cada responsabilidad:	
Nombre:	SetCurrentState
Descripción:	Esta función tiene la responsabilidad de cambiar el estado actual.
Nombre:	SetGlobalState
Descripción:	Esta función tiene la responsabilidad de cambiar el estado global.
Nombre:	SetPreviousState
Descripción:	Esta función tiene la responsabilidad de cambiar el estado anterior.
Nombre:	Update
Descripción:	Esta función tiene la responsabilidad de ejecutar el estado global y el estado actual.
Nombre:	HandleMessage
Descripción:	Esta función tiene la responsabilidad de pasar los mensajes recibido a los estados en ejecución para ver si pueden ser manejados por estos.
Nombre:	ChangeState
Descripción:	Esta función tiene la responsabilidad de cambiar el estado anterior por el estado actual y el estado actual por el estado pasado por parámetro.
Nombre:	RevertToPreviousState
Descripción:	Esta función tiene la responsabilidad de cambiar el estado actual por el estado anterior
Nombre:	isInState

Descripción:	Esta función tiene la responsabilidad de determinar si la máquina de estados tiene como estado actual al estado pasado por parámetro
Nombre:	CurrentState
Descripción:	Esta función tiene la responsabilidad de devolver el estado actual.
Nombre:	GlobalState
Descripción:	Esta función tiene la responsabilidad de devolver el estado global.
Nombre:	PreviousState
Descripción:	Esta función tiene la responsabilidad de devolver el estado anterior.
Nombre:	GetNameOfCurrentState
Descripción:	Esta función tiene la responsabilidad de el nombre del estado actual.

Tabla 58. Descripción de la clase StateMachine

Nombre: Telegram	
Tipo de clase : Entidad	
Atributo	**Tipo**
Sender	int
Receiver	int
Msg	int
ExtraInfo	double
DispatchTime	void

Tabla 59. Descripción de la clase Telegram

Nombre: MessageDispatcher	
Tipo de clase : Controladora	
Atributo	**Tipo**
PriorityQ	set<Telegram>
Para cada responsabilidad:	
Nombre:	Discharge
Descripción:	Esta función es utilizada por DispatchMsg o DispatchDelayedMessages y tiene la responsabilidad de enviar un mensaje a la entidad especificada en el telegrama como destino del mismo.
Nombre:	Instance
Descripción:	Esta función tiene la responsabilidad de devolver una instancia estática de esta clase.
Nombre:	DispatchMsg
Descripción:	Esta función tiene como responsabilidad crear un telegrama y enviarlo a la

	entidad especificada como destino y si el tiempo de retardo es nulo si no guardarlo en la cola con prioridad.
Nombre:	DispatchDelayedMessages
Descripción:	Esta función tiene la responsabilidad de enviar todos los mensajes de la cola con prioridad que ya vencieron su tiempo de retardo.

Tabla 60. Descripción de la clase MessageDispatcher

Nombre: FrameCounter	
Tipo de clase :	
Atributo	**Tipo**
m_lCount	long
m_iFramesElapsed	int
Para cada responsabilidad:	
Nombre:	Instance
Descripción:	Esta función tiene la responsabilidad de devolver una instancia estática de esta clase.
Nombre:	Update
Descripción:	Esta función tiene la responsabilidad de incrementar el contador y la cantidad de frames transcurridos.
Nombre:	GetCurrentFrame
Descripción:	Esta función tiene la responsabilidad de devolver el valor del contador.
Nombre:	Reset
Descripción:	Esta función tiene la responsabilidad de hacer cero el contador.
Nombre:	Start
Descripción:	Esta función tiene la responsabilidad de hacer cero la cantidad de frames transcurridos.
Nombre:	FramesElapsedSinceStartCalled
Descripción:	Esta función tiene la responsabilidad de devolver la cantidad de frames transcurridos.

Tabla 61. Descripción de la clase FrameCounter

3.4. Diagrama de Despliegue

Todos los componentes del software a desarrollar están empaquetados en un solo nodo físico. Este diagrama consta de una sola computadora personal, por lo tanto se considera que no es necesario reflejarlo en este informe.

3.5. Diagramas de Componentes

A continuación se presentan los diagramas de componentes que agrupan los archivos de código fuente que fueron usados para la realización de este sistema. Se han definido tres paquetes de componentes para hacer más fácil su representación, estos coinciden con la organización por módulos del sistema.

3.5.1. Paquete "Módulo Máquina de Estados"

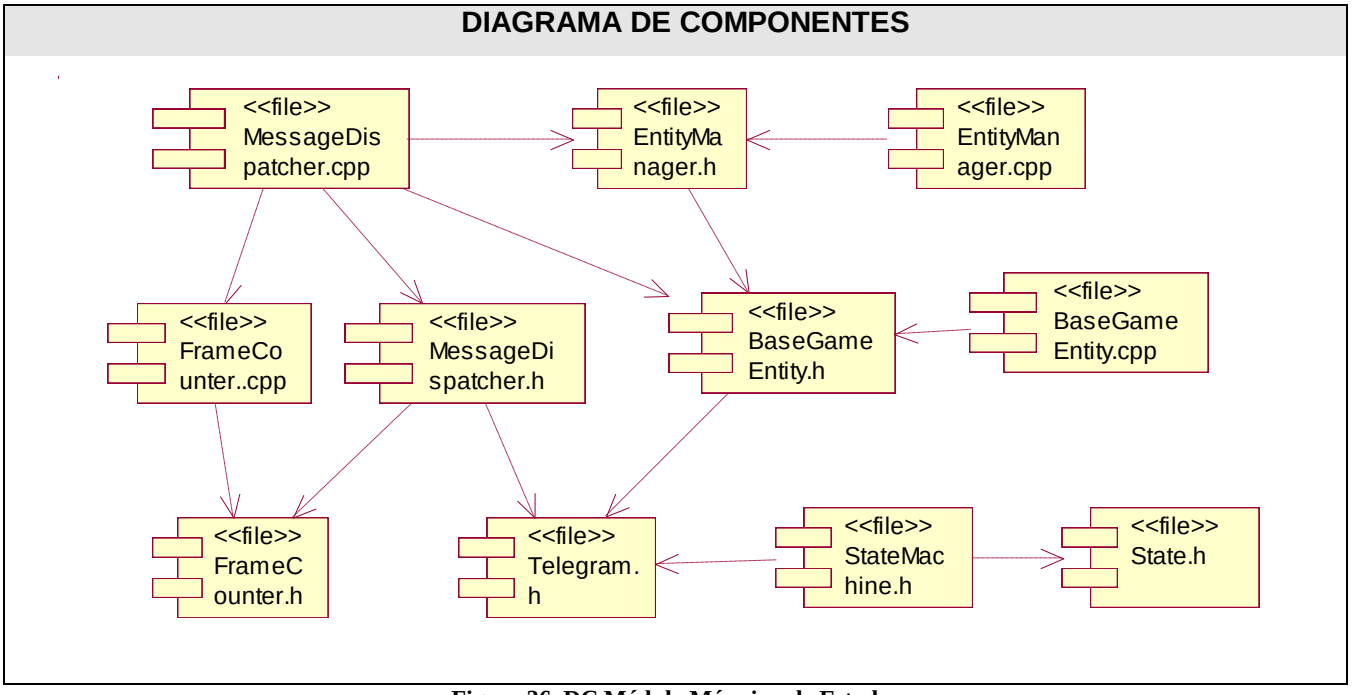

Figura 36. DC Módulo Máquina de Estados

3.5.2. Paquete “Módulo Lógica Difusa”

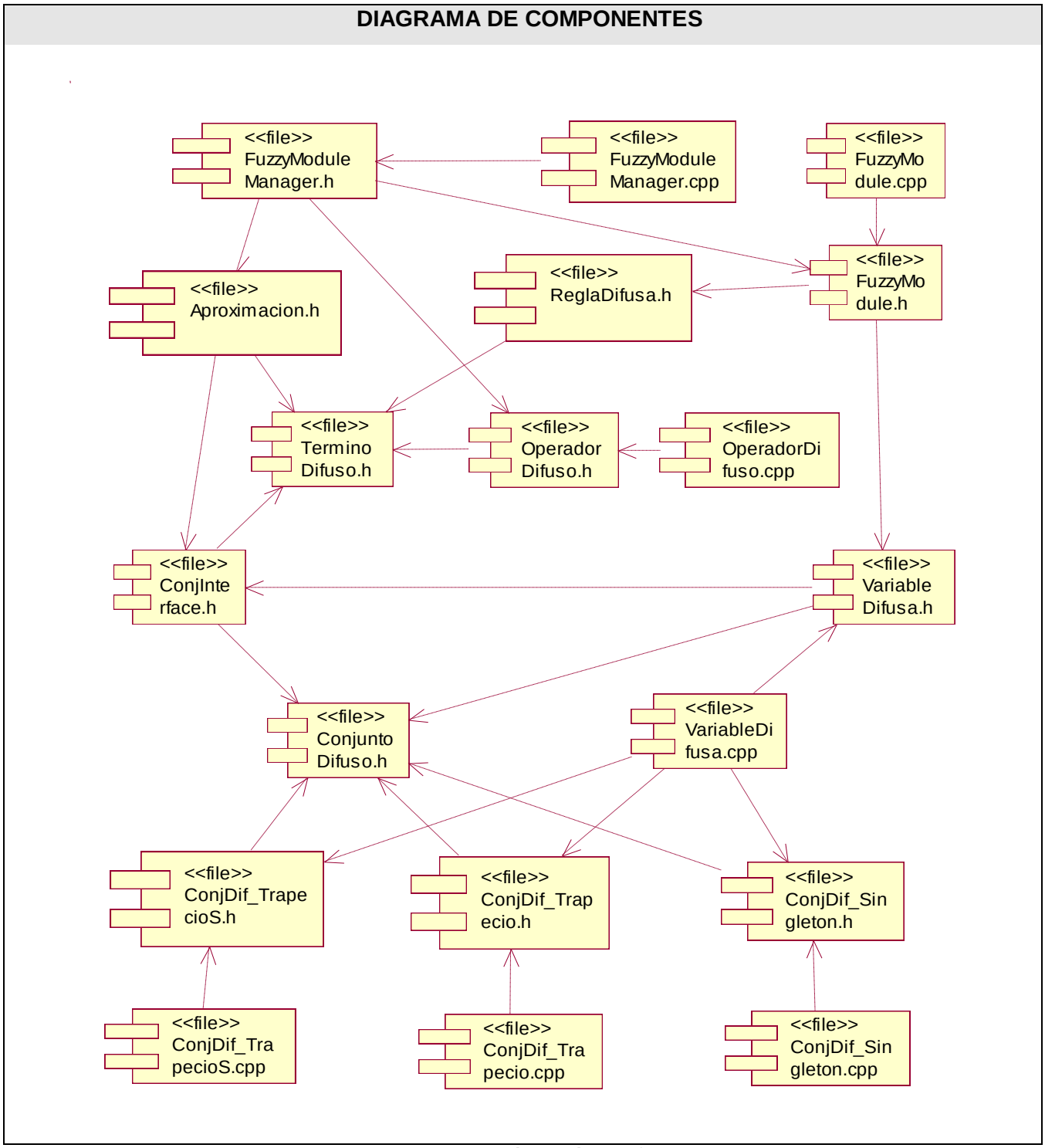

Figura 37. DC Módulo Lógica Difusa

3.5.3. Paquete “Módulo Parser”

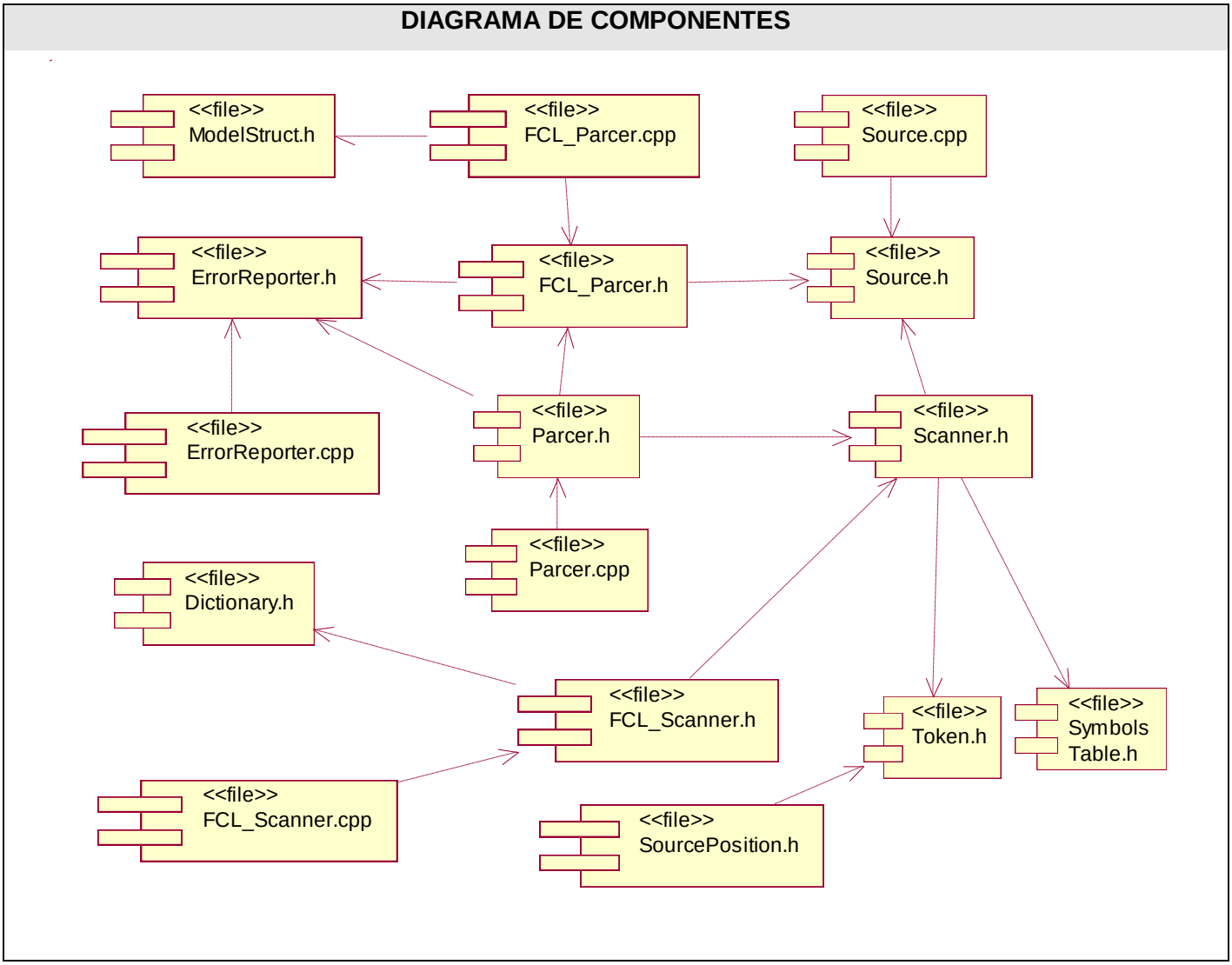

Figura 38. DC Módulo Parser FCL

CONCLUSIONES

Para el cumplimiento de los objetivos de este trabajo y los requisitos del cliente, se hizo una revisión de la literatura mediante la cual se extrajo la información relevante sobre el tema a investigar. Se complementó el anterior paso con el estudio de las diferentes herramientas de toma de decisiones, principalmente las de lógica difusa existentes a nivel internacional. En el estudio se analizaron las principales características de dichas herramientas. A partir de esta investigación se pudo constatar cómo desde los primeros años de la industria de los videojuegos la inteligencia artificial y particularmente el proceso de toma de decisiones de los personajes que forman parte de estos juegos, han sido un centro de especial atención de los especialistas y programadores, llegándose a incursionar en los últimos tiempos en técnicas más sofisticadas y no deterministas. Esta búsqueda dio paso a la propuesta de las características técnicas de esta solución.

Se realizó la captura de requisitos, tanto funcionales como no funcionales, y la agrupación de los primeros en los casos de uso del sistema. Seguidamente se transitó por las fases de análisis y diseño, producto de lo cual se obtuvieron las clases que estructuran el sistema. Se implementaron los casos de uso, generando el código fuente de la aplicación y los componentes físicos de la arquitectura.

Se obtuvo un sistema de toma de decisiones para agentes virtuales de un videojuego con la ventaja de poder ser utilizado con cualquier motor gráfico, independientemente del sistema operativo y la plataforma de desarrollo. Con la creación de este producto la Facultad da un paso en aras de lograr un sistema de IA genérico que contribuya a una mayor calidad de los videojuegos desarrollados por el Polo Productivo de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas.

RECOMENDACIONES

Finalizada la ejecución de este trabajo, y logrados los objetivos planteados se recomienda:

- Incrementar la compatibilidad del sistema con el estándar FCL.
- La integración del sistema con un sistema de IA general.
- La aplicación del sistema en los proyectos productivos de videojuegos de la Facultad.
- La incorporación de otras técnicas de tomas de decisiones tanto deterministas como no deterministas como redes neuronales, redes bayesianas y árboles de decisiones para hacer el sistema más robusto.
- El desarrollo de un subsistema que permita el aprendizaje de los agentes virtuales.

REFERENCIAS BIBLIOGRÁFICAS

1. UBP - Diplomatura en Programación de Videojuegos. [En línea] http://www.ubp.edu.ar/pagina2772.html.

2. **Wiles, Janet y Johnson, Daniel.** *Computer Games With Intelligence 1.*

3. **Duch i Gavaldà, Jordi y Tejedor Navarro, Heliodoro.** *Inteligencia Artificial.* s.l. : Universitat Oberta de Catalunya.

4. **Millington, Ian.** *Artificial Intelligence for Games.*

5. **Schwab, Brian.** *AI Game Engine Programming.* Hingham, Massachusetts : Charles River Media, Inc, 2004.

6. **Nareyek, Alexander.** *AI, in Computer Games.* Carnegie Mellon University : s.n., 2004.

7. **Rabin, Steve.** *A1 Game Programming Wisdom 2.* Massachusetts : Charles River Media, 2004.

8. DevMaster.net-Unreal Engine 3. [En línea] http://www.devmaster.net/engines/engine_details.php?id=25.

9. DevMaster.net-Quest3d. [En línea] http://www.devmaster.net/engines/engine_details.php?id=80.

10. DevMaster.net -3D Game Studio. [En línea] http://www.devmaster.net/engines/engine_details.php?id=67.

11. DevMaster.net -Artificial Engines. [En línea] http://www.devmaster.net/engines/engine_details.php?id=282.

12. DevMaster.net -RealmForge. [En línea] http://www.devmaster.net/engines/engine_details.php?id=68.

13. DevMaster.net -Elemental Engine II. [En línea] http://www.devmaster.net/engines/engine_details.php?id=572.

14. Free Fuzzy Logic Library. [En línea] http://ffll.sourceforge.net/.

15. XFuzzy 3.0. [En línea] http://www.imse.cnm.es/Xfuzzy/Xfuzzy_3.0/index_sp.html.

16. JFuzzyLogic. [En línea] http://jfuzzylogic.sourceforge.net/.

17. Free Fuzzy Logic Library. [En línea] http://ffll.sourceforge.net/fcl.htm.

18. **Trujillo Rivero, Andy y Márquez Rodríguez, Marvyn Amado.** *Definición del comportamiento de carros autónomos en un videojuego de carreras empleando redes neuronales artificiales.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.

19. **Prevot Urgellés, Yunerkis y Herrera Bacallao, Arianna.** *APLICACIONES DE LAS REDES NEURONALES EN ENTORNOS VIRTUALES.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.

20. **Martínez Labaut, Tamara y Antúnez Ojeda, Yaima.** *PROPUESTA DE TÉCNICAS DE APRENDIZAJE DE ELEMENTOS VIRTUALES.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.

21. **Valdés Pera, Lisandra y Sánchez Berrillo, Haydee.** *Propuesta de Técnica para la Comunicación entre Agentes Virtuales.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, 2008.

22. **Buckland, Mack.** *Programming Game AI by Example.* s.l. : Wordware Publishing, 2005.

23. wiktionary.org. [En línea] http://es.wiktionary.org/wiki/videojuego.

24. Red Científica. [En línea] http://www.redcientifica.com/doc/doc199903310001.html

25. Procesado y Optimización de Espectros Raman mediante Técnicas de Lógica Difusa: Aplicación a la identificación de Materiales Pictóricos. [En línea]. http://www.tesisenxarxa.net/TESIS_UPC/AVAILABLE/TDX-0207105-105056

26. Soluciones y Propuestas RATIONAL –Software y Hardware IBM. [En Línea]. http://www.rational.com.ar/herramientas/rup.html

27. MSDN Home Page. [En Línea] http://msdn.microsoft.com/es-ar/vstudio/products/bb931214.aspx

28. **Martín del Brío, Bonifacio y Sanz Molina, Alfredo.** Redes Neuronales y Sistemas Difusos.

Alfaomega Ra-Ma.

29. **Lima, Yanaris Bravo y García Miniet, Raiza Sareda.** *Aplicación de la Lógica Difusa al módulo de corte del Simulador Quirúrgico de la Universidad de las Ciencias Informáticas.* Ciudad de la Habana : Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, 2008.

BIBLIOGRAFÍA

Gutiérrez, Prof. José Manuel. *Sistemas Expertos Basados en Reglas.* s.l. : Dpto. de Matemática Aplicada. Universidad de Cantabria.

Robert St. Amanr, R. Michael Young. *Artificial Intelligence and Interactive Entertainment.*

Wexler, James. *Artificial Intelligence in Games:A look at the smarts behind Lionhead Studio's "Black and White" and where it can and will go in the future.* s.l. : University of Rochester.

M. Bourg David, Seeman Glenn. AI For Game Developers. O'Reilly Media, 2004

Gamma, Erich, y otros. *Design Patterns. Elements of Reusable Object-Oriented Software.*

ANEXOS

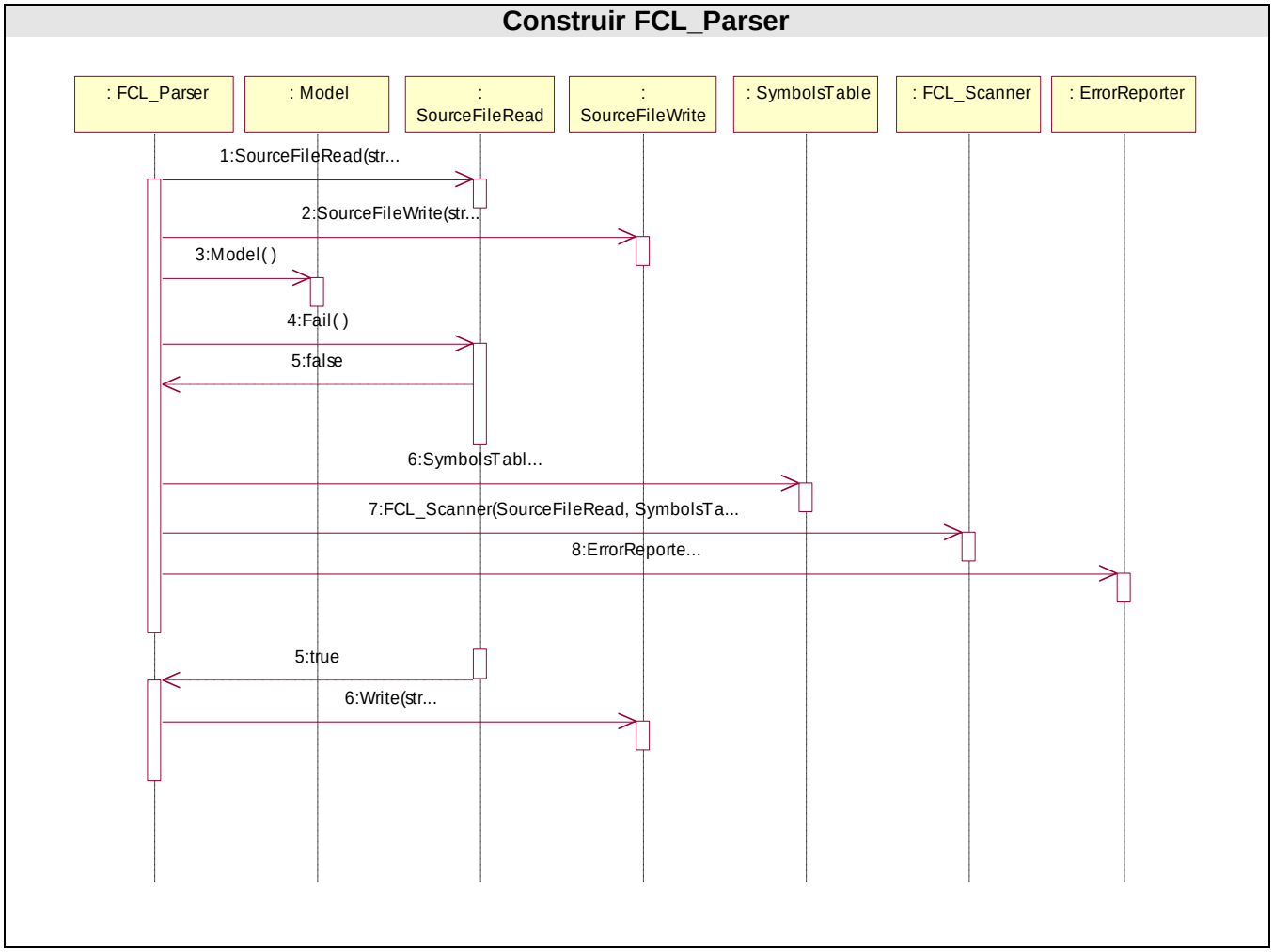

Figura 39. Diagrama de Secuencia CU Adicionar Entidad

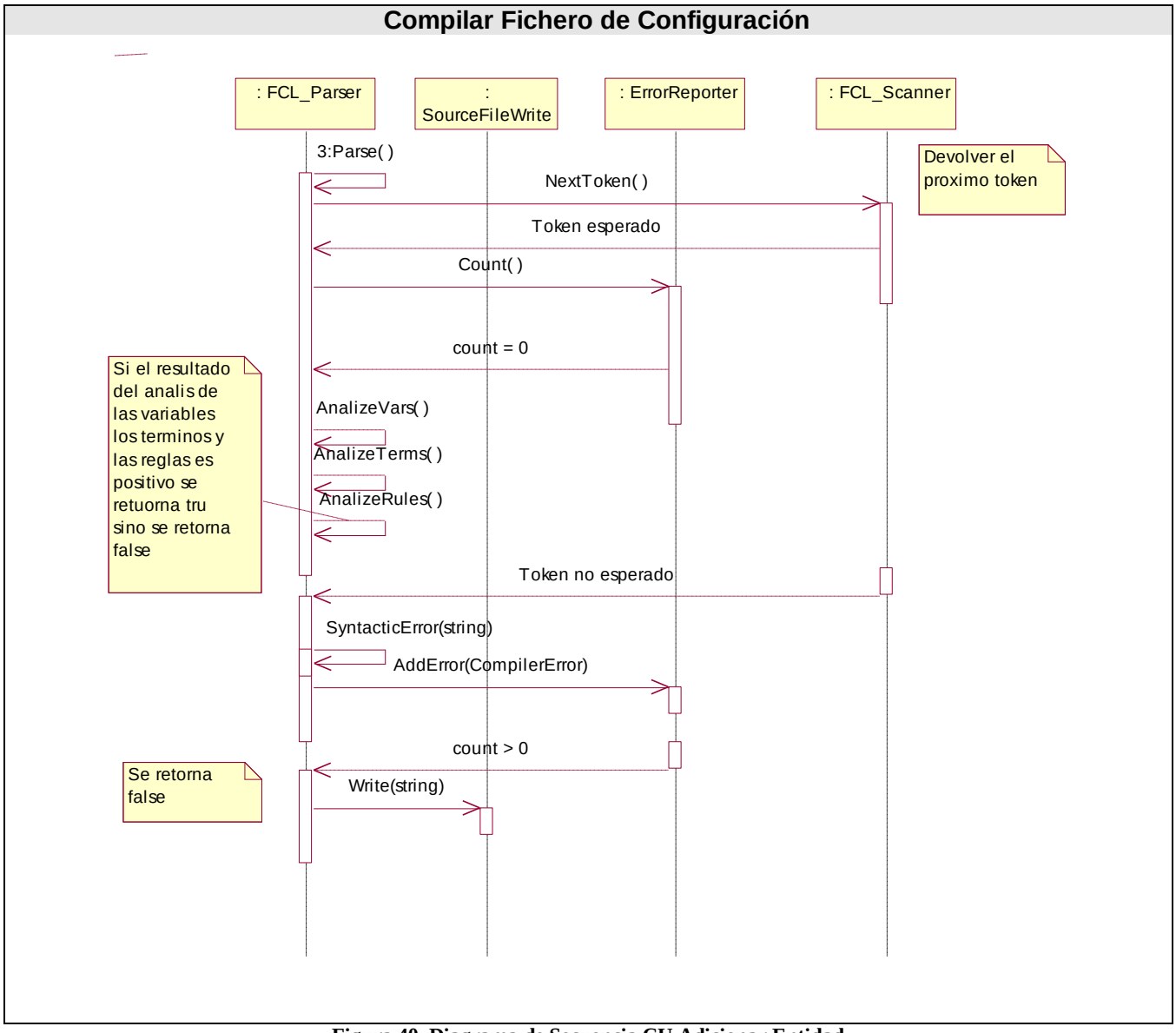

Figura 40. Diagrama de Secuencia CU Adicionar Entidad

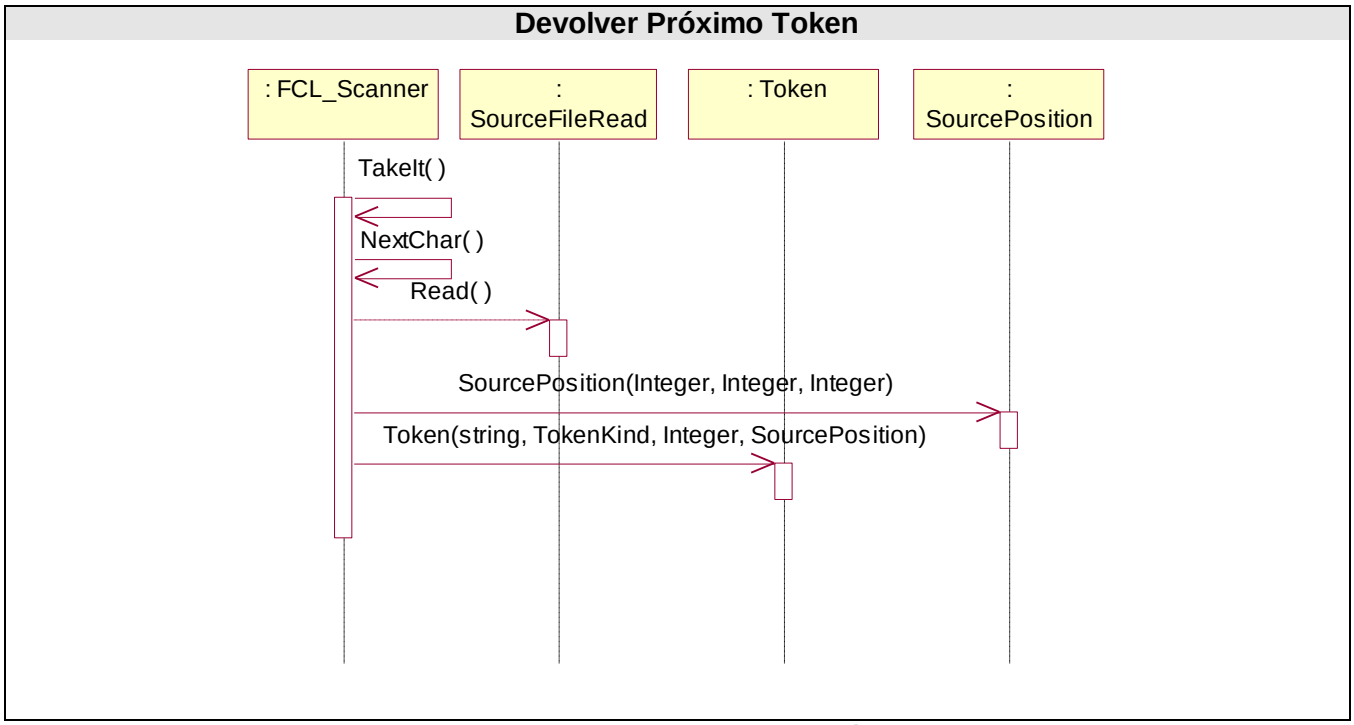

Figura 41. Diagrama de Secuencia Devolver Próximo Token.

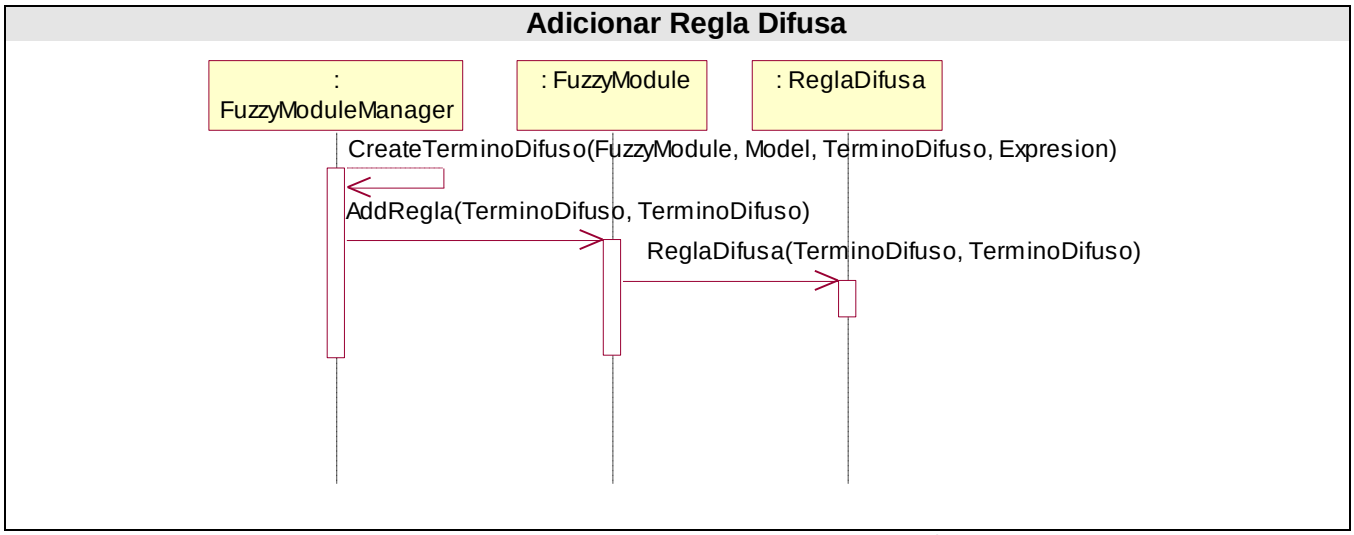

Figura 42. Diagrama de Secuencia Adicionar Regla Difusa.

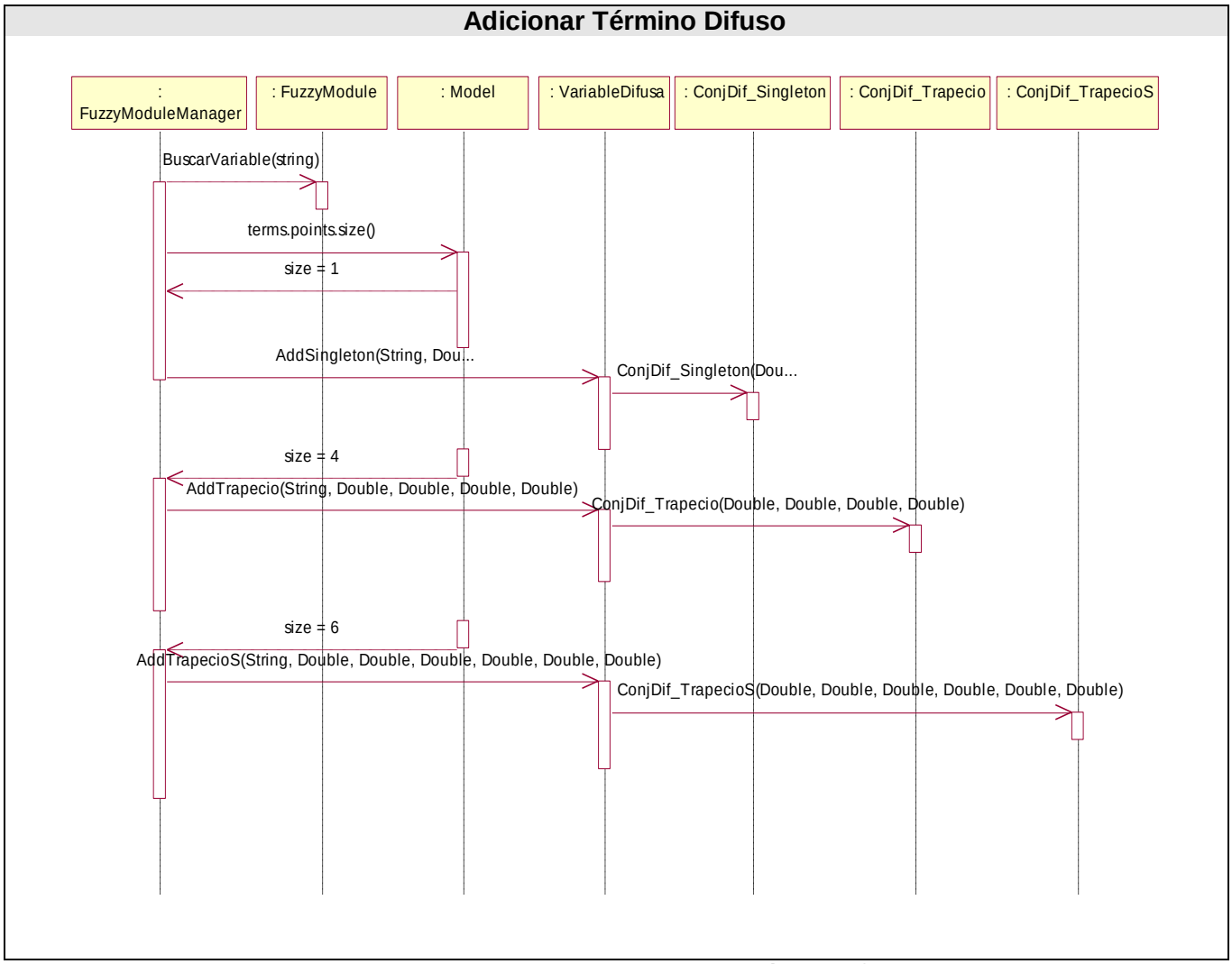

Figura 43. Diagrama de Secuencia Adicionar Término Difuso.

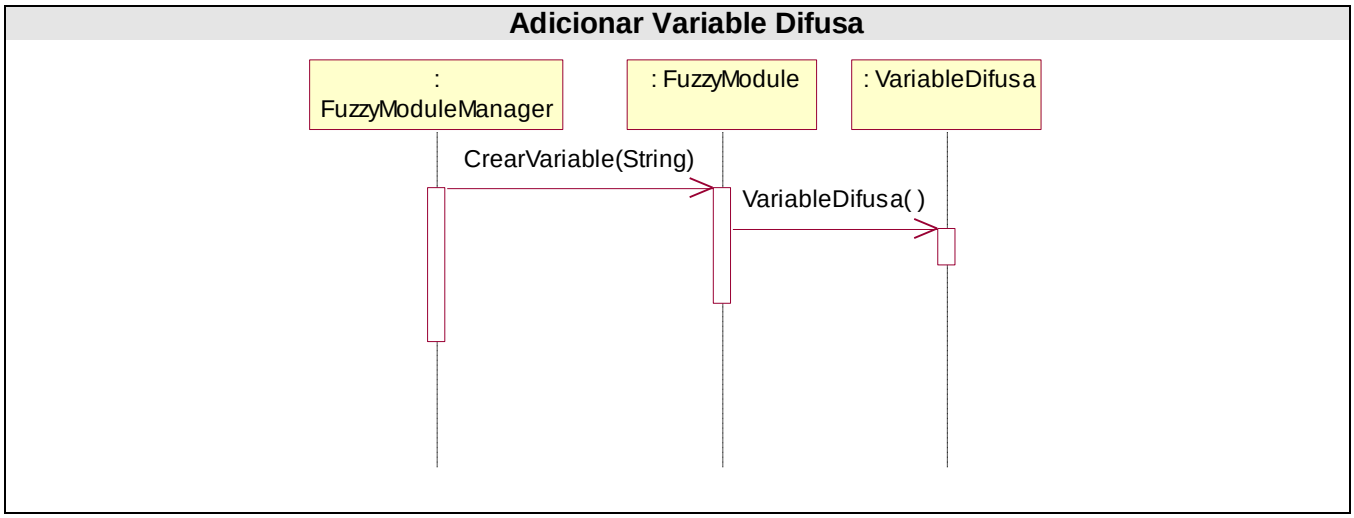

Figura 44. Diagrama de Secuencia Adicionar Variable Difusa.

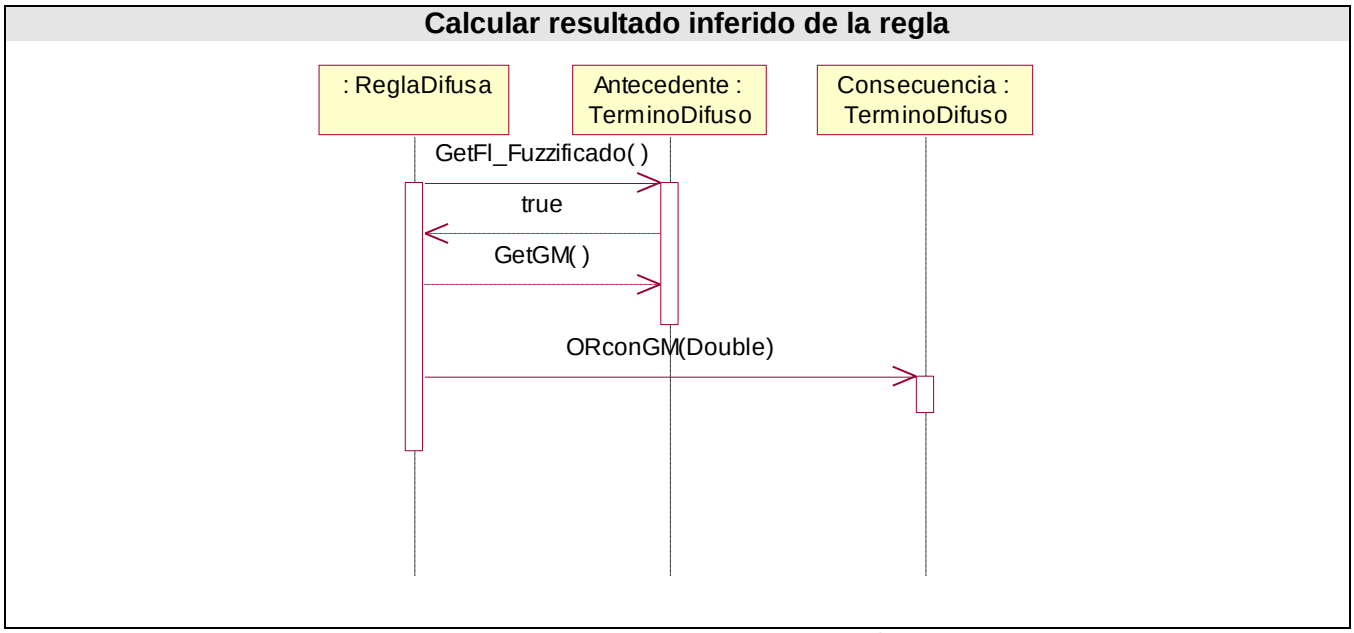

Figura 45. Diagrama de Secuencia Calcular resultado inferido de la regla.

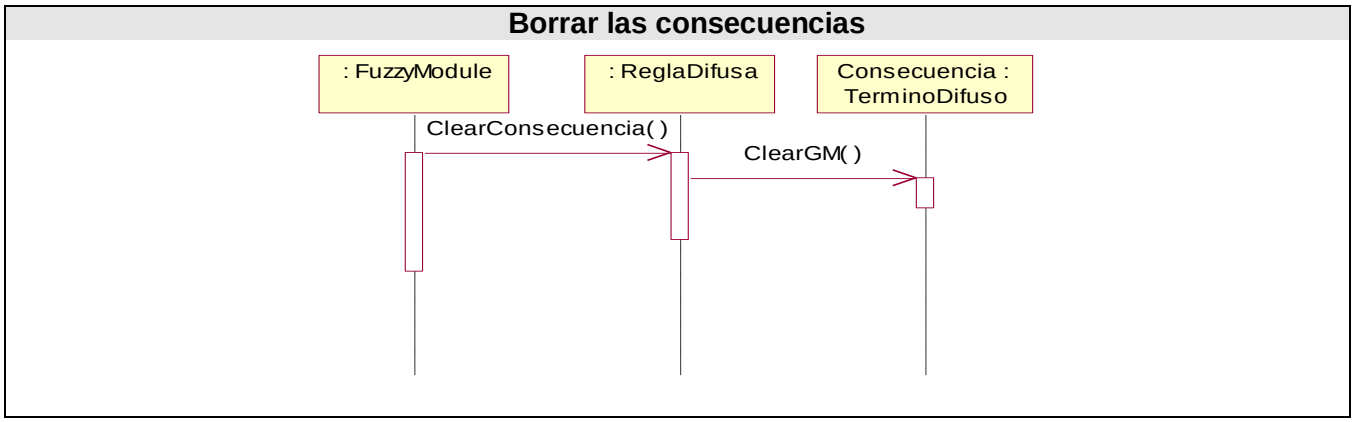

Figura 46. Diagrama de Secuencia Borrar las consecuencias.

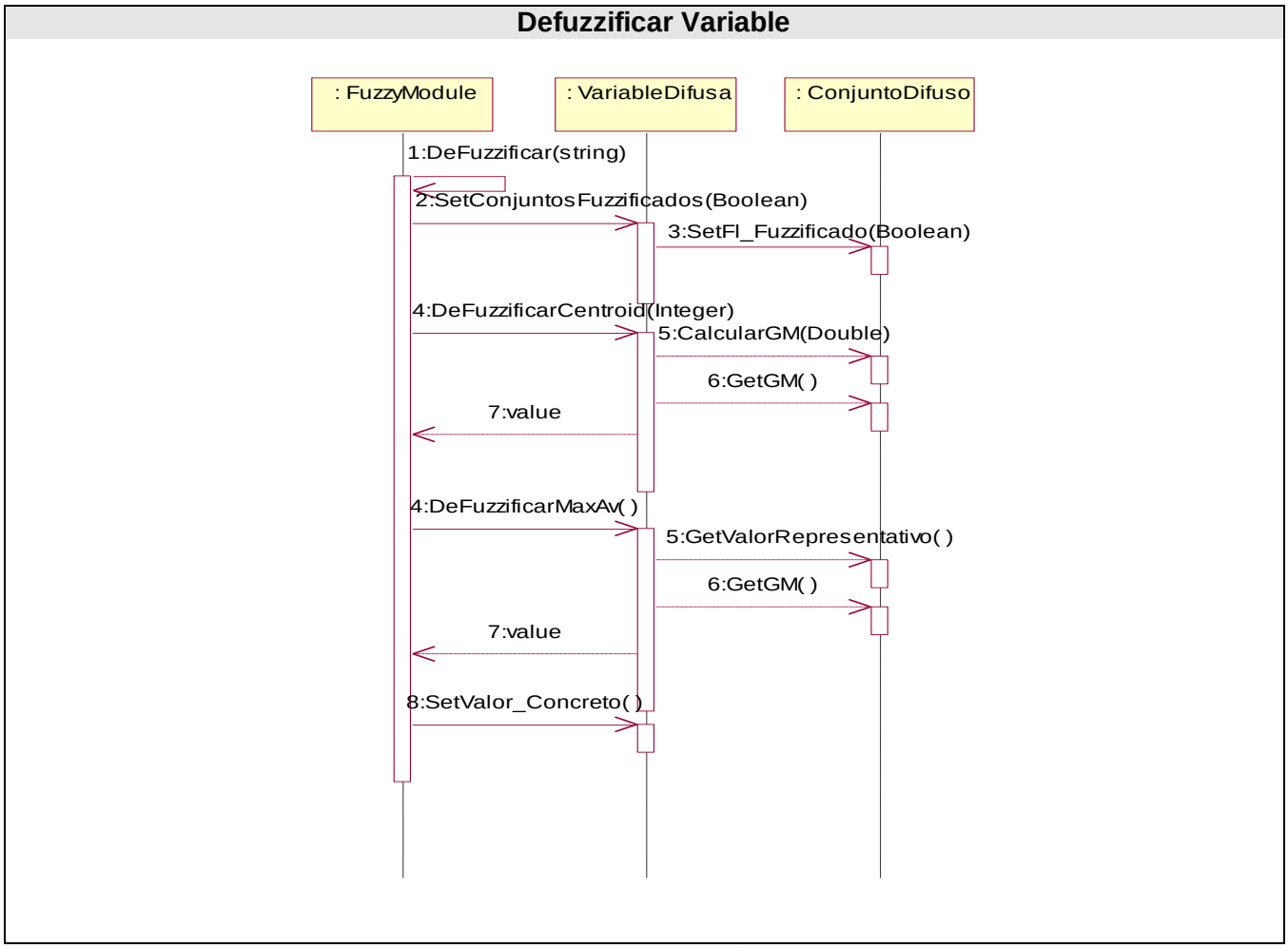

Figura 47. Diagrama de Secuencia Defuzzificar Variable.

1 GLOSARIO

[1] **STK**: Siglas de *SceneTooKit*, es una Biblioteca Gráfica de Tiempo Real propia para los proyectos del Polo de Realidad Virtual, con utilidades para el desarrollo de aplicaciones de realidad virtual. Es un conjunto de clases y funciones para el trabajo básico en sistemas de realidad virtual y gráfica por computadora en general.

[2] **Motor Gráfico**: Entendemos como motor gráfico una API que nos permite trabajar con gráficos desde un nivel alto, abstrayendo todo el proceso de dibujo y centrándose en la gestión de los elementos. Una colección de estructuras, funciones y algoritmos utilizados para visualizar objetos tridimensionales en una pantalla bidimensional.

[3] **NPC**: En los juegos y videojuegos de rol, personaje no controlado por el jugador (por las siglas en inglés: *No Player Character*).

[4] **Editor de Nivel**: (También conocidos como editor de campañas, mapas o escenarios) es una aplicación de software usada para el diseño de niveles, campañas o escenarios de un videojuego. En algunos casos los creadores de videojuegos entregan una versión oficial del editor de niveles en sus juegos.

[5] **Fuzzificación**: Para cada variable numérica de entrada se obtiene su valor de pertenencia a cada uno de los conjuntos difusos en que se divide el universo de discurso.

[6] **Defuzzificación**: Dado los valores de pertenencia ”M(y)” de todos los conjuntos que forman parte del universo de discurso de una variable se obtiene su valor numérico.

[7] **International Electrotechnical Commission**: La Comisión Electrotécnica Internacional (CEI o IEC, por sus siglas del idioma inglés International Electrotechnical Commission) es una organización de normalización en los campos eléctrico, electrónico y tecnologías relacionadas. Numerosas normas se desarrollan conjuntamente con la ISO (normas ISO/IEC). A la CEI se le debe el desarrollo y difusión de los estándares para algunas unidades de medida, particularmente el gauss, hercio y weber; así como la primera propuesta de un sistema de unidades estándar, el sistema Giorgi, que con el tiempo se convertiría en el sistema internacional de unidades.

[8] **Patrones GOF**: Patrones de diseño de software recogidos en el libro Design Patterns escrito por el grupo Gang of Four (GoF) compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogen 23 patrones de diseño comunes.

[9] **Patrón State**: Patrón de diseño perteneciente a los patrones GOF. Este patrón permite a un objeto modificar su comportamiento a medida que su estado interno va cambiando.