Universidad Carlos III de Madrid

Escuela politécnica Superior

Ingeniería Técnica en Informática de Gestión

PROYECTO DE FIN DE CARRERA

Motor de razonamiento basado en reglas para el simulador empresarial SIMBA

Alumno: Daniel Sánchez Cisneros

Director: Fernando Fernández Rebollo

Año: 2009

AGRADECIMIENTOS

A mi tutor el Dr. Fernando Fernández Rebollo por la inestimable ayuda prestada y la rapidez con la que ha contestado a todos mis correos.

A todos los profesores que me han ayudado durante el tiempo que se han prolongado mis estudios universitarios.

A mi padre, mi madre y mis hermanos por aguantar los buenos y, sobretodo, los malos momentos vividos durante todos estos años.

A todos los compañeros con los que he tenido el privilegio de compartir el entorno del Laboratorio PLG, en especial a Francisco Javier, que han hecho posible, gracias a su ayuda siempre cordial y dispuesta, que este proyecto haya podido llevarse a buen término.

Por último, aunque no menos importante, a mi pareja, amigos y compañeros de carrera que me han aportado ánimos y comprensión desde siempre, y que suponen una parte importantísima de las motivaciones que me mueven.

A todos ellos: MUCHAS GRACIAS.

Índice General

Capítulo 1

Introducción

La informática es la disciplina que nació de la gestión automática de la información para facilitar al ser humano el tratamiento de datos en procesos complejos y en muchas ocasiones en un ámbito científico. Es por ello que el hombre sintió la necesidad de automatizar procesos lógicos referentes al cálculo de datos que eran repetitivos.

Esta automatización de la información dio sus primeros pasos en el s. XVII cuando, entre muchos otros hitos, Blaise Pascal creó la primera máquina calculadora mediante engranajes en 1642.

Más adelante la automatización de cálculo llegó al punto de poder representar la información de algoritmos matemáticos simples y el análisis de condiciones para el posterior cómputo de datos dentro de algoritmos matemáticos con aportes como el álgebra de Boole en 1854 que serviría posteriormente para la estructura del primer computador de Von Newman en 1949. Sin embargo no toda la informática se centra en la computación de algoritmos matemáticos o científicos, sino que hay ramas que investigan la automatización del comportamiento y lo que es aún más complicado, la inteligencia.

La inteligencia artificial es la ciencia dentro del ámbito de la informática que investiga sobre estructuras lógicas, algoritmos de resolución, heurísticas de búsqueda y metodologías del pensamiento para construir sistemas capaces de resolver problemas complejos de manera automática. En ocasiones la inteligencia artificial investiga sobre patrones en el razonamiento así como el comportamiento y la inteligencia del ser humano para adaptarlas a la resolución de estos problemas.

La representación del conocimiento es un aspecto fundamental para realizar sistemas eficientes que resuelvan problemas complejos debido a que existen varias posibilidades dentro de la inteligencia artificial. Escoger la técnica adecuada es crucial. En este caso se investiga profundamente en los sistemas de producción basados en reglas. Esto quiere decir que se utiliza un conjunto de reglas para definir el comportamiento que debe desarrollar un sistema.

Una de las aplicaciones más importantes de la Inteligencia Artificial es realizar simuladores. Los simuladores son una herramienta que permite analizar y estudiar áreas del conocimiento que son peligrosas (como por ejemplo el comportamiento de una explosión volcánica en la base de su

cráter y su expansión de lava) o simplemente imposible de conocer empíricamente (cosas que pasarán en el futuro). Para dar una estimación de la solución a estos problemas o una aproximación al análisis para un estudio se pueden utilizar algunas técnicas y metodologías de la inteligencia artificial que tratan sobre sistemas basados en el conocimiento y más específicamente de sistemas basados en reglas.

Dentro de todos los tipos de simuladores que existen, hay una serie de simuladores dedicados al entorno económico y áreas del mercado empresarial. Estos simuladores normalmente están compuestos por un gran número de variables que definen un entorno o situación. Dada una situación a simular, los directivos de mercado deben tomar unas decisiones que repercutirán en el entorno simulado (afectarán sobre cada variable en su justa medida y por tanto presentarán un estado del entorno diferente). De esta manera los directivos son capaces de obtener una estimación sobre qué podría pasar en el futuro si tomasen una decisión u otra como, por ejemplo, dedicar más fondos en un periodo contable al departamento de I+D.

En estos últimos años se ha dado un crecimiento bastante amplio del número de simuladores empresariales creados para preveer posibles cambios en el mercado empresarial y poder hasta cierto punto, anticiparse a los cambios estimados. Siempre hay que respetar la salvedad de que los resultados previstos son una estimación.

En el año 2009 una empresa de base tecnológica creada por miembros del grupo de investigación INNOVATIC de la Universidad Autónoma de Madrid, llamada Simuladores Empresariales S.L. desarrolló en colaboración con el grupo *Planning and Learning Group* del departamento de Informática de la Universidad Carlos III de Madrid un simulador empresarial llamado SIMBA. Este simulador tiene como finalidad realizar predicciones de mercado, evaluando cada empresa en función de una toma de decisiones realizada por sus directivos. Finalmente se puntúa cada empresa y se elabora un ranking de mercado en el que figuran todas las empresas o equipos junto con su entorno empresarial y de mercado.

Según se especifica en el artículo de su desarrollo de trabajo *[4]*, el sistema SIMBA está compuesto por varios módulos, agentes inteligentes y un motor de razonamiento. Este último es clave para el cálculo del entorno de mercado y ranking de una empresa o equipo. El objetivo sobre el que se centra este proyecto es crear un nuevo motor de razonamiento basado en reglas que sustituya al motor vigente.

La motivación principal de crear un motor de razonamiento basado en reglas es que tiene grandes ventajas, como poder modificar su patrón de comportamiento en tiempo de ejecución añadiendo o eliminando reglas. De esta manera eliminamos la necesidad de parar el motor de razonamiento, modificarlo, compilarlo y volverlo a ejecutar, con la perdida de tiempo (traducida en dinero para las empresas) que esta parada conlleva.

Otra razón que motiva este proyecto es que las reglas empresariales están en continua evolución y cambio. Esto se convierte en un inconveniente para el motor vigente porque está programado íntegramente en c++, que es difícil de actualizar y versionar. Es decir, que si las reglas empresariales cambian con respecto a las ya implementadas entonces tenemos un problema porque son difíciles de modificar o actualizar en el motor.

Debido a sus características intrínsecas, el lenguaje c++ se aleja bastante del lenguaje natural o matemático (relativamente). Esto dificulta que un usuario no familiarizado con la programación pueda actualizar los patrones o fórmulas que determinan el comportamiento del arbitraje.

En resumen, esto es todo lo que motiva a desarrollar un motor de razonamiento basado en reglas. Y para llevarlo a cabo es fundamental apoyarse en un lenguaje de programación que permita los sistemas de producción basados en reglas y orientados a objetos. Por esta razón se eligió el lenguaje de programación CLIPS. También hay que mencionar que cabía la posibilidad de utilizar otros lenguajes de programación como Drools o Prolog entre otros, que también soportan sistemas de reglas.

Sin embargo los sistemas basados en el conocimiento también tienen sus inconvenientes. Uno de estos problemas es que estos motores no tienen un orden de ejecución establecido, sino que van ejecutando las reglas que se puedan ejecutar en cada momento. Esto es perjudicial en el sentido de que no se puede controlar de manera sencilla el orden de ejecución, y por lo tanto es difícil realizar un seguimiento de los cálculos que el motor de razonamiento ha realizado. Por otro lado, aunque sea difícil realizar un seguimiento del comportamiento del motor no quiere decir que sea imposible. Una manera sería realizando un historial de las reglas que se han ejecutado. El motor de razonamiento basado en reglas puede seguir una serie de estrategias de búsqueda para ejecutar cada regla posible en cada momento, como por ejemplo las estrategias de ejecución en profundidad, etc. Por lo que concluimos que el orden de ejecución del motor de razonamiento no es aleatorio ni mucho menos.

Otro de los inconvenientes clásicos de los sistemas basados en reglas es el tiempo de ejecución que necesitan. El tiempo de ejecución de un motor de razonamiento implementado con un lenguaje estructurado está condicionado por bastantes factores que determinan en gran medida el tiempo de ejecución, como por ejemplo el número de accesos a la Base de Datos y el tipo de accesos que realiza. Como veremos más adelante, en este proyecto se han tenido muy en cuenta estos factores y se ha realizado un acceso a la Base de Datos diferente al que había ya implementado, lo cual ha sido notablemente diferenciador en cuanto a tiempo.

Para este fin de eficiencia también habrá que tener en cuenta la estructura de datos del nuevo motor de razonamiento. Por ello se intenta realizar en CLIPS una nueva estructuración de los datos basada en la que utiliza el simulador empresarial SIMBA, pero aportando enfoques diferentes para filtrar datos innecesarios. Esta nueva visión de la información prevé acortar al máximo posible el tiempo de cómputo del mismo.

Con respecto a la información tratada en este proyecto no hay que olvidar un aspecto fundamental como es la privacidad. Esto quiere decir que en el presente documento muchos de los datos utilizados han sido omitidos para cumplir el acuerdo de confidencialidad con la empresa Simuladores Empresariales S.L.

Después de crear el nuevo motor e integrarlo en el sistema SIMBA, se han realizado pruebas de rendimiento y eficiencia. Estos datos se han comparado con los obtenidos en el antiguo motor del sistema. Para ello hay que tener en cuenta varios factores diferenciales de cada motor, como por ejemplo la manera dinámica en la que cada motor accede a la base de datos, los resultados reportados, etc.

Para realizar este proyecto se han seguido una serie de pasos que explicamos ordenadamente en este documento de esta manera: en el capítulo 2 se presentan los orígenes de la problemática, así como la evolución de los simuladores. También se explica el sistema SIMBA debido a la gran importancia que toma en este proyecto (el nuevo motor de razonamiento que se desarrolle será integrado en dicho sistema). En este capítulo además se explican los Sistemas Basados en el Conocimiento y finalmente el lenguaje utilizado para la implementación del nuevo motor de razonamiento. En el capítulo 3 se exponen los objetivos generales y específicos que dan sentido a la realización de este proyecto. En el capítulo 4 se explica la memoria del trabajo realizado: preparar, gestionar, estructurar y desarrollar el nuevo motor de razonamiento basado en reglas. En el capítulo 5 se presenta el resultado que ha dado el motor de razonamiento y su rendimiento en comparación con el motor antiguo ante una amplia batería de pruebas que se le han realizado. En el capítulo 6 se explica la gestión del proyecto mediante técnicas de Ingeniería del Software y la misma gestión de proyectos. En el capitulo 7 se exponen las conclusiones obtenidas de este proyecto y en el capítulo 8 se proponen futuras líneas de trabajo de este estudio. En el Anexo I se muestra información útil para entender la distribución del motor de razonamiento basado en reglas y la información privada del proyecto, incluyendo un manual de usuario o ejemplo de ejecución de la aplicación.

Capítulo 2

Estado de la cuestión

En este capitulo se va a realizar una exposición de los conocimientos que constituyen la base de este proyecto. Para ello comenzaremos en el apartado 2.1 realizando una introducción al estado de la cuestión desde la historia sobre los simuladores empresariales. Posteriormente en los apartados 2.2, 2.3 y 2.4 expondremos el simulador empresarial SIMBA, su arquitectura funcional y su arquitectura software. En el apartado 2.5 explicaremos los sistemas basados en el conocimiento. Finalmente en el apartado 2.6 explicamos el funcionamiento de CLIPS y los sistemas basados en reglas.

2.1. Simuladores empresariales

Los juegos de simulación tienen sus orígenes en los juegos de mesa, datados sus inicios en China alrededor del año 3.000 a.c., donde el *Majong* entre otros muchos fue precedido hasta los modernos juegos como el *Monopoly* (1934). Sin embargo los juegos empresariales modernos o de simulaciones en administraciones empresariales no hicieron su puesta en escena hasta 1955. En 1956, apareció el primer simulador de gestión: un simulador de gestión de decisiones, desarrollado por la *American Management Association*, fue usado con fines educativos en la universidad de Washington en 1957, seguida de cerca por un juego de gestión empresarial desarrollado por *McKinsey&CO* que fue el primero en ser usado en una clase. La evolución en el uso y tipo de simuladores gestores ha sido revisada en diferentes estudios. Por supuesto, los avances en IT (Information Technology) han mejorado la tecnología del juego, dando una respuesta rápida y mejorando la usabilidad y accesibilidad.

Los resultados educativos resaltan el éxito de estos productos software. Los simuladores empresariales mejoran la velocidad de transferencia de teoría a práctica, la cual reduce costes y ahorra tiempo de entrenamiento. Esto representa un cambio en los modelos educativos, haciendo más rápido el cambio de conexiones entre los conocimientos o fundamentos teóricos y las decisiones de acción y prácticas empresariales. Este cambio en los modelos educativos ha demostrado su eficacia en la consecución de procesos envueltos en la toma de decisiones. Como resultado de esto tenemos las mejoras de las capacidades, competencias, habilidades y cualidades de los estudiantes que han utilizado estas modernas metodologías educativas. Los resultados

destacan que la mayor mejora de los estudiantes se produce en las capacidades de gestión de organizaciones, creatividad, criterio, interacción, discusión, y adquisición de conocimiento.

Existe alguna controversia como qué tipo de simulaciones mejoran el conocimiento. En primer lugar, hay problemas de interrelaciones en las áreas funcionales de la compañía, en las variables de flujo y acción, en relaciones no lineales, en información cualitativa y en resultados estructurales. Esto es debido a que el simulador apunta sólo a un área comercial de la compañía. El propósito de un enfoque integrado (sistema de aproximación dinámica) ayuda a resolver esta clase de problemas. Los simuladores integran la toma de decisiones automática en sistemas dinámicos, considerando que la toma de decisiones no tiene consistencia ni persistencia en las estructuras que no usan algoritmos complejos. Los simuladores no calculan la solución óptima (no hay solución óptima en economía), ni tampoco intentan obtener toda la información para apoyar la toma de decisiones debido a que no todas las decisiones son racionales matemáticamente y no son proporcionales cuantitativamente con el desarrollo.

Por otro lado, estos aspectos han generado discusiones sobre el tipo de simuladores de juego empresarial a usar: el modelo “Black box” donde se oculta información o por otro lado el “White box” que usa la información de manera transparente. El modelo Black box está compuesto por estados (determinado por atributos) donde la toma de decisiones no usa toda la información ni tampoco se realizan los cálculos o soluciones a los problemas complejos que no son enteramente racionales. Los juegos empresariales con White box ofrecen al usuario herramientas de diagramas de casos y proporcionan una metodología de resolución de cálculos, pero tienen la desventaja de que sólo pueden usar ecuaciones y relaciones simples porque las relaciones y modelos matemáticos complejos no sería inteligibles para el usuario.

2.2. SIMBA

SIMBA (SIMulation in Business Administration) [4] es el trabajo de una empresa spin-off de la Universidad Autónoma de Madrid llamada Simuladores Empresariales S.L. en colaboración con la Universidad Carlos III de Madrid. Podríamos definir SIMBA como un programa computacional Web que simula el entorno de una serie de mercados. Este programa es el resultado de cerca de 20 años de experiencia en la simulación empresarial ambas en el entorno de la educación universitaria y el entrenamiento ejecutivo, que emula la realidad empresarial usando las mismas variables, relaciones y eventos presentes el mundo empresarial. El propósito es proveer a los usuarios de una visión integrada de la compañía usando la misma base de reglas, relaciones y mercados dinámicos presentes en la gestión empresarial, simplificando la complejidad y destacando el contenido y los principios de mayor valor pedagógico.

SIMBA tiene muchas características de gran valor. En primer lugar, es un simulador empresarial enmarcado en un entorno competitivo, donde o bien un equipo de participantes puede competir contra otras compañías gestionadas automáticamente por el simulador mediante agentes inteligentes, o bien donde múltiples equipos de participantes compiten entre ellos. Es un simulador multifuncional porque está encaminado a las áreas funcionales principales de la compañía. Además es interactivo porque permite a los participantes comunicarse tanto con el simulador como con otros participantes de la simulación y al mismo tiempo es muy versátil porque tiene diferentes niveles de dificultad, adaptándolo al nivel de conocimiento de gestión empresarial de los participantes.

No hay límite en el número de usuarios. Permite la diversificación de productos, mercados, segmentos de clientes y tecnologías. Además, como es un sistema Web, los usuarios pueden acceder directamente desde cualquier ordenador conectado a Internet, lo cual elimina la necesidad de instalar la aplicación en un entorno local de IT. Además se puede hacer funcionar en cualquier plataforma o Sistema Operativo, por no mencionar que puede ser usado en cualquier lugar y en cualquier dispositivo que tenga acceso a la Web. También se puede personalizar SIMBA en cuanto a lengua, configuración, moneda y entorno socioeconómico según la demanda. SIMBA es compatible con un gran número de gráficos, hojas de datos y utiliza los principales indicadores a seguir tanto por las compañías como por los mercados, ediciones de mercado, organizaciones económicas, eventos empresariales que podrían alterar el mercado dinámico y el ranking. También posee módulos de análisis para evaluar objetivamente la gestión del rendimiento del equipo.

La evolución de la tecnología de la información y comunicaciones, junto con la nueva demanda de unos modelos de educación superior (adaptación a diferentes tipos de usuario, enseñanza a distancia, etc.) justifican el desarrollo de nuevas herramientas de entrenamiento, como es el caso

de SIMBA. Centrándonos más específicamente en el avance pedagógico de los simuladores empresariales, debemos destacar la "inmersión de aprendizaje" como una característica de esta metodología. Esto significa que el estudiante usa técnicas y conceptos adquiridos durante su entrenamiento, y son necesarios para analizar, tomar decisiones y evaluar resultados. Así se obtiene una experiencia parcial y un conocimiento reforzado. Esta característica se acentúa con la posibilidad de elegir un agente inteligente específico para actuar como competidor, adaptando el mercado y el comportamiento del competidor a las características del grupo de estudio.

SIMBA ha sido usado en escuelas empresariales y programas de entrenamiento en los que la administración empresarial es casi el núcleo de contenidos. En este contexto SIMBA ha desempeñado dos roles alternativos. Por un lado ha sido una actividad integrada para los participantes (un entorno empresarial neutral con el que interactúan, se conocen unos a otros y realizan diferentes roles profesionales) y por otro lado es una herramienta para afianzar conceptos y poner en práctica el conocimiento y las técnicas estudiadas en la teoría.

Existen también beneficios derivados de usar simuladores en la educación empresarial. Algunos beneficios están relacionados a los objetivos del aprendizaje técnico, y otros beneficios tienen que ver con el desarrollo de capacidades y competencias en los sistemas de educación modernos y configuración empresarial.

Además si dejamos a un lado la parte económica o empresarial y miramos la parte técnica de este simulador, veremos que se puede obtener un gran provecho en el ámbito de la educación informática. Uno de los principales inconvenientes de las asignaturas muy específicas es encontrar una plataforma de desarrollo adecuada para realizar prácticas. Este es el caso de la asignatura Aprendizaje Automático [5] impartida por el departamento de informática de la Universidad Carlos III de Madrid. En esta asignatura es importante aprender no sólo cómo se aplican las técnicas, sino cómo funcionan los algoritmos que implementan esas técnicas. Entonces aprovechando el uso de agentes inteligentes en el simulador empresarial SIMBA, así como la alta carga de competitividad que lleva intrínseca, se ha utilizado este simulador para las prácticas de la asignatura de Aprendizaje Automático. Más específicamente la construcción de agentes virtuales capaces de tomar decisiones en el entorno de SIMBA es una tarea lo suficientemente interesante y sencilla para los alumnos como para plantearla durante el curso de Aprendizaje Automático. Por este motivo, las prácticas que se han diseñado van dirigidas a que al final del curso los alumnos cuenten con sus propios agentes inteligentes.

Todas estas características hacen de SIMBA una herramienta única capaz de crear una inmersión en un entorno de aprendizaje eficaz para alcanzar los objetivos educativos de diferentes áreas.

2.3. Arquitectura funcional de SIMBA

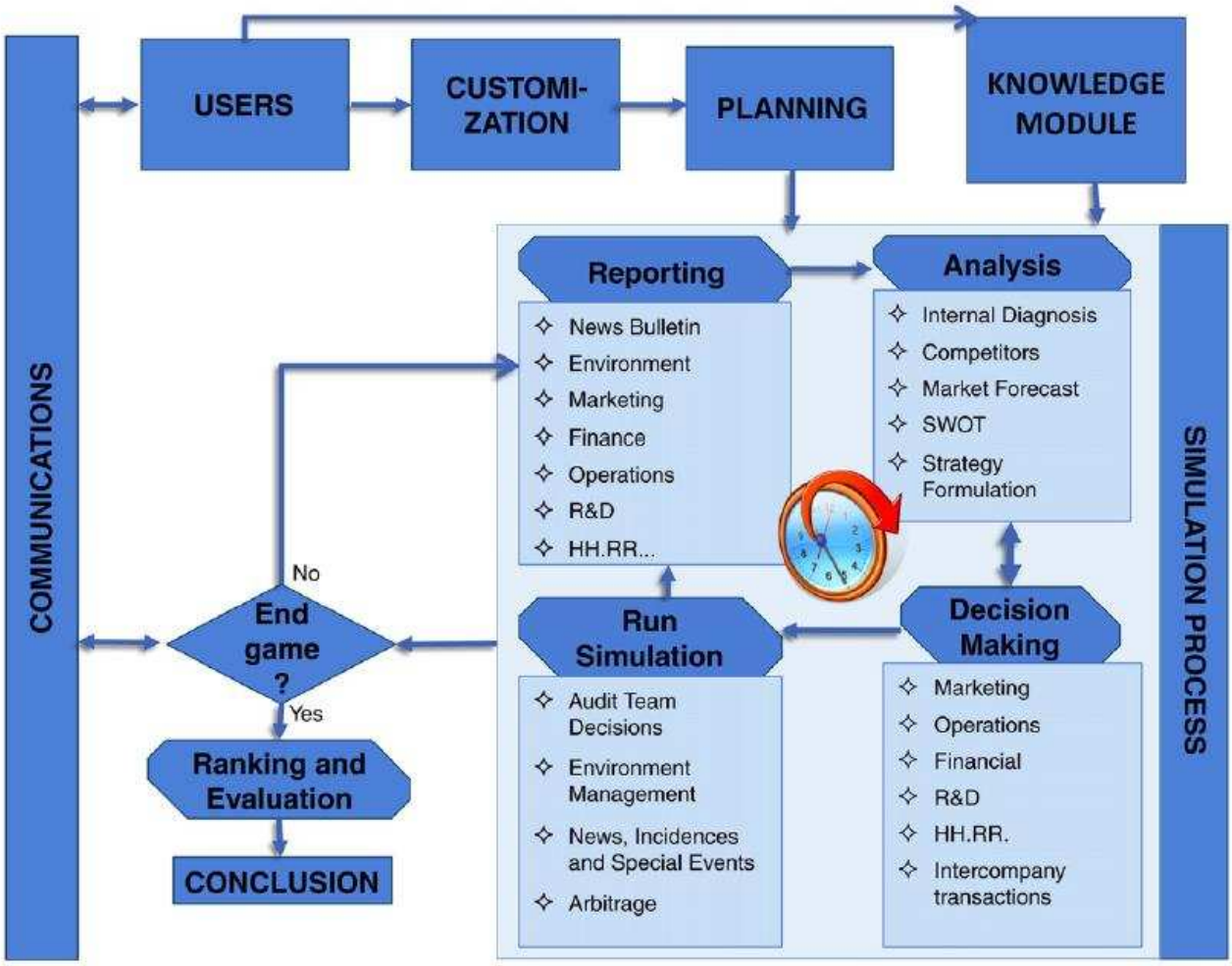


Figura 2: Arquitectura de SIMBA [4]

La figura 1 da una perspectiva general de la estructura de SIMBA, mostrando los diferentes módulos de operaciones desde el punto de vista del usuario. La estructura consiste en grupos de subsistemas especialmente diseñados para proveer un alto estándar de versatilidad y usabilidad. Estas dos últimas características son necesarias para la participación y la administración de la simulación. Solamente se describen los módulos cuyas funcionalidades son de alguna manera singulares. El módulo de usuario permite al administrador de sistemas definir perfiles de usuario para satisfacer las necesidades de las instituciones que usarán el simulador. Este módulo permite la creación de clientes, administradores, instructores o asistentes y participantes. SIMBA también provee la funcionalidad de ser ejecutada automáticamente respondiendo a las especificaciones predefinidas, sin necesitar el control del administrador.

Los perfiles de instructor y asistente pueden ser diseñados para trabajar como un gran grupo de enseñanza –como una competición a nivel internacional- para adaptarse a las regulaciones académicas encontradas en la enseñanza superior.

El módulo de personalización cubre el rango de funciones cuyo objetivo es crear variables del entorno como los productos, entornos geográficos y económicos, situación inicial de las compañías, nuevos mercados, etc. Todo ello ajustado a la moneda y lengua de la competición. Toda esta información permite la generación de simuladores a la medida de las necesidades de los participantes. Este módulo tiene una muy importante e interesantes capacidad de adaptación, que es la clave para explotar el potencial de SIMBA.

El módulo de planificación permite al instructor organizar y definir las características de cada simulación. Éste crea equipos formados por participantes, determinando el número de decisiones a tomar y su planificación. También define el número de mercados y compañías en cada mercado, formado por equipos a compañías. Después el módulo tiene que determinar cómo los mercados estarán estructurados en términos de concurrencia, lengua, productos, entornos económicos y parámetros de entorno.

El módulo de conocimiento tiene como objetivo proveer un soporte on-line y tutorías a los participantes.

El proceso de simulación consiste en cuatro grandes módulos especializados, que configuran la dinámica de la competición. Cada uno de estos módulos soluciona una tarea: el módulo de presentación de informes (reporting) se encarga de dar información inicial a los participantes. El módulo de analisis (Analysis) analiza la información a través de técnicas de diagnóstico. El módulo de toma de decisiones (decision making) sirve para que o bien los participantes tomen decisiones o bien los agentes inteligentes generen las decisiones automáticamente. Finalmente el módulo de arbitraje (run simulation) sirve para controlar la dinámica de la competición.

Este proceso de arbitraje comienza cuando el módulo de presentación de informes genera toda la información que el equipo necesita saber sobre su compañía, sus competidores y el entorno de competición, etc. Después, en el módulo de análisis los participantes del equipo aplican sus habilidades de gestión y negociación para definir la estrategia que seguirán en el futuro y obtener una buena posición competitiva de su empresa. Para ello decidirán aspectos como la tecnología de análisis que usarán, el estilo de toma de decisiones que realizarán, etc. Después se toman las decisiones apropiadas usando la interfaz SIMBA en el módulo de toma de decisiones.

Los módulos de análisis y toma de decisiones forman el escenario principal donde los participantes desarrollan sus roles como directivos de toma de decisiones. Los participantes hacen uso de aproximadamente 25 variables de mercado, organizadas en áreas funcionales. La media estimada del tiempo de toma de decisiones es de 2 horas por cada mercado.

Una vez que los participantes han realizado la toma de decisiones, comienza el funcionamiento del módulo de arbitraje. Este módulo tiene como objetivo controlar la dinámica de la competición y la interacción entre los equipos y los instructores. Se ejecuta siempre que un periodo de toma de decisiones termina, ajustándose al calendario planificado y verificando que todos los equipos han tomado sus decisiones. Este módulo también permite a los instructores seleccionar noticias e incidencias, algunas de las cuales pueden ser creadas en el módulo de personalización. Esto puede alterar la dinámica del mercado, lo cual adapta al simulador a las habilidades y capacidades de los participantes. El proceso de arbitraje incluye los resultados y la toma de decisiones del periodo previo, los parámetros del entorno económico general, las características geográficas de cada mercado, etc. Como resultado de este proceso de arbitraje, el motor de razonamiento del simulador empieza a calcular y generar los resultados de salida del periodo que estamos arbitrando. Este proceso se describirá en detalle en próximos puntos.

El proceso de arbitraje descrito, en el que intervienen los cuatro módulos de la arquitectura funcional de SIMBA, es llevado a cabo por todas las compañías de un mercado. Después, con los resultados obtenidos, todos los mercados entran en competición a través de SIMBA.

Una vez que todo ha terminado, volvemos a comenzar otro ciclo de arbitraje donde el módulo de presentación de informes facilitará a los participantes los resultados del arbitraje que se acaba de realizar en el periodo anterior. Después se volverá a utilizar el módulo de análisis y toma de decisiones, etc. Esta consecución de arbitrajes continuará hasta el final de la competición, donde se realizará un ranking en el módulo de evaluación.

Con respecto a la elaboración de este ranking, SIMBA ofrece un listado de equipos formados por participantes en cada simulación. Para elaborar el ranking, se usa un procedimiento con varios criterios sobre una serie de indicadores (principalmente económicos, comerciales, financieros y de gestión de las magnitudes). SIMBA permite al instructor dar una ponderación a cada uno de los indicadores, dependiendo de la importancia que quiera dar a cada uno. Además cuando hay múltiples mercados en una simulación, se aplica un procedimiento de evaluación más complejo. SIMBA tiene un ranking consolidado o puntuación de las empresas, que el equipo gestiona durante la competición. Esta puntuación es calculada, consolidando el posicionamiento de cada unidad estratégica (productos, mercados, clientes o tecnologías alternativas) en cada uno de los siguientes criterios: posicionamiento por tamaño de la empresa (cuyo peso está en concordancia con su clave económica, financiera y gestionada por sus indicadores), posicionamiento por diversificación o riesgo de cartera empresarial y posicionamiento de acuerdo al valor de mercado o ciclo de vida de la industria que desarrolla.

Estos tres criterios se han consolidado en un ranking de ponderación simple, cuyo resultado está disponible para los equipos que lo necesiten, dependiendo de criterio del instructor. La ponderación puede ser cambiada según el criterio del instructor:

Ranking = 0,2*TpuntROI + 0,15*TpuntROE + 0,1*TpuntProductividad

+ 0,1*TpuntCosteProducto + 0,1*TpuntMercadoCompartido

+ 0,1*TpuntDebt + 0,25*TpuntValorDeMercado

Este procedimiento da la oportunidad al instructor de abrir un grupo de discusión sobre la gestión y comportamiento de mercado de los diferentes equipos, valorando los éxitos, fracasos, logros y errores junto con la dinámica de la competición que se discute. Finalmente los ganadores y perdedores pueden ser identificados. Esto hace que el módulo de ranking sea muy útil, así como una herramienta distintiva para los propósitos educativos.

Es importante recordar que la motivación del proyecto es la usabilidad que nos proporcionan los sistemas basados en reglas. Esto quiere decir que si queremos modificar los algoritmos que rigen el arbitraje (porque las reglas del mercado empresarial hayan cambiado y las que tenemos hayan quedado obsoletas), o si queremos modificar los algoritmos de evaluación que determinan el ranking, basta con modificar las reglas del nuevo motor basado en reglas que se desarrolla en este proyecto. Este cambio de comportamiento se realizará de manera inmediata sin necesidad de parar el sistema para compilar.

Además las reglas están más próximas al lenguaje natural, lo que facilita y acerca el motor de razonamiento a usuarios que no tienen porqué estar familiarizados con la programación (no es necesario personal especializado).

Una vez que se ha explicado el proceso general que sigue el arbitraje de un periodo, ahora vamos a profundizar un poco más específicamente el proceso de arbitraje que se realiza en el módulo de arbitraje. Este proceso de arbitraje se puede ver en la figura 3.

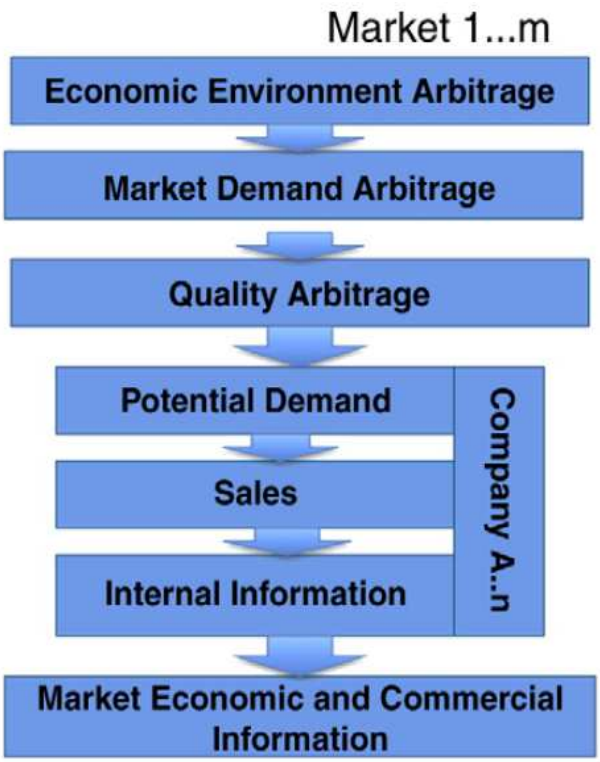


Figura 3: Orden de trabajo en la ejecución de un arbitraje [4]

Para comenzar el proceso de arbitraje dentro del módulo de arbitraje, el primer paso determinar el entorno económico. Este entorno económico tiene como objetivo definir el contexto en el que el arbitraje de mercado tendrá lugar. Las variables que definen este contexto son entre otras la inflación, índice de confianza de los clientes, ratios de interés de los préstamos financieros y disponibilidad de factores de producción (mano de obra y tecnología, etc.). Estas variables se comportan según una base aleatoria, que comienza desde una situación inicial generada bajo las especificaciones consistentes del instructor, y evolucionan según el valor de la inflación dado.

Después pasamos a determinar la demanda del mercado. La demanda de mercado tiene como objetivo generar un total de la demanda para un producto, que el mercado absorberá bajo las condiciones de entorno económico actuales. La generación de demanda está fuertemente influenciada por el entorno económico y las condiciones de mercado anteriores. La demanda generada determinará la evolución futura del comportamiento del mercado, en la medida en que las compañías son capaces de cumplir sus expectativas de mercado. Las variables más significativas que afectan al potencial de la demanda son entre otros las proyecciones de demanda a largo plazo, las ventas de la estacionalidad y el esfuerzo de marketing de cada empresa en el mercado.

Una vez determinada la demanda de mercado, pasamos a determinar los factores de calidad. La calidad del mercado evalúa entre otros factores la comercialización, la manufacturación industrial y el esfuerzo invertido en I+D de cada empresas para asignar a las compañías diferentes segmentos de mercado (Alto, Medio o Bajo). Este proceso es llevado acabo por una función compleja que involucra a 14 variables y 38 parámetros, que se ponderan entorno a un lapso de tiempo medio de 3 años. Una expresión simplificada del algoritmo que ejecuta el arbitraje de calidad es el siguiente:

Calidad(t) = f [(pcalidad(t-1, t-2, t-3)*calidad(t-1, t-2, t-3)),

(pr&d(t, t-1, t-2, t-3)*I+D(t, t-1, t-2, t-3)),

ppublcity(t, t-1, t-2, t-3), *(Publicidad(t, t-1, t-2, t-3)),

(pdscto(t, t-1, t-2, t-3)*dcto(t, t-1, t-2, t-3)),

(pentr(t, t-1, t-2, t-3)*entr(t, t-1, t-2, t-3))]

El nivel de calidad de cada empresa también tiene en cuenta la consistencia o volatilidad a lo largo del tiempo de las decisiones referentes a I+D, las política de marketing o las políticas de gestión. El análisis de estos factores es llevado a cabo por un grupo logístico de reglas internas del proceso de calidad.

Los niveles de calidad obtenidos por cada empresa van a determinar en gran medida el siguiente paso del proceso de arbitraje: la demanda potencial de cada empresa. La demanda potencial obtenida de cada empresa, da al mercado una figura de la demanda esperada en relación con la capacidad real de respuesta de cada compañía.

Después se calcula el descuento de la empresa, que consiste en la comparación de la demanda de mercado para la empresa con su inventario de stock acabado real. Si la demanda supera al stock, entonces la diferencia será asignada a otros competidores con stock disponible y un posicionamiento en el mercado similar.

Este paso completa la evaluación de mercado y crea un procedimiento para generar información interna sobre otras áreas de la empresa. Después el proceso de arbitraje culmina con un proceso de generación de informes, que involucra tanto información interna como externa (producción, RRHH, estados financieros, información de mercado, stock de mercado, etc.).

El objetivo de este proyecto es implementar en CLIPS todo este proceso de arbitraje que se ha explicado. Además, la explicación del proceso de arbitraje sirve para entender el funcionamiento de los algoritmos desarrollados dentro del motor de razonamiento basado en reglas.

2.4. Arquitectura software de SIMBA

Un aspecto muy importante de la arquitectura de SIMBA es que las personas pueden interactuar con agentes autónomos. Esta sección describe como SIMBA puede ser usado para la investigación en aplicaciones de la IA, especialmente en sistemas multiagentes dentro de la administración empresarial. Los sistemas multiagentes pueden mejorar el futuro de la inteligencia empresarial mediante las nuevas tecnologías y los sistemas de apoyo de decisiones. SIMBA puede incluir varios agentes autónomos para realizar el rol de equipos competidores, basados en la investigación de la toma de decisiones de los equipos humanos. Últimamente se han realizado muchos estudios para mejorar la complexión y efectividad de estos agentes inteligentes.

La arquitectura de SIMBA permite el uso de diferentes participantes, incluyendo tanto agentes software como participantes humanos. Los participantes intervienen en la simulación a lo largo de todos los periodos de la competición. En todos los arbitrajes, los participantes (software y humanos) reciben un estado del entorno económico y realizan una toma de decisiones. Una vez que todos los equipos han realizado la toma de decisiones, se realiza el arbitraje. La figura 3 muestra la arquitectura software del simulador empresarial desde la perspectiva de un sistema multiagente.

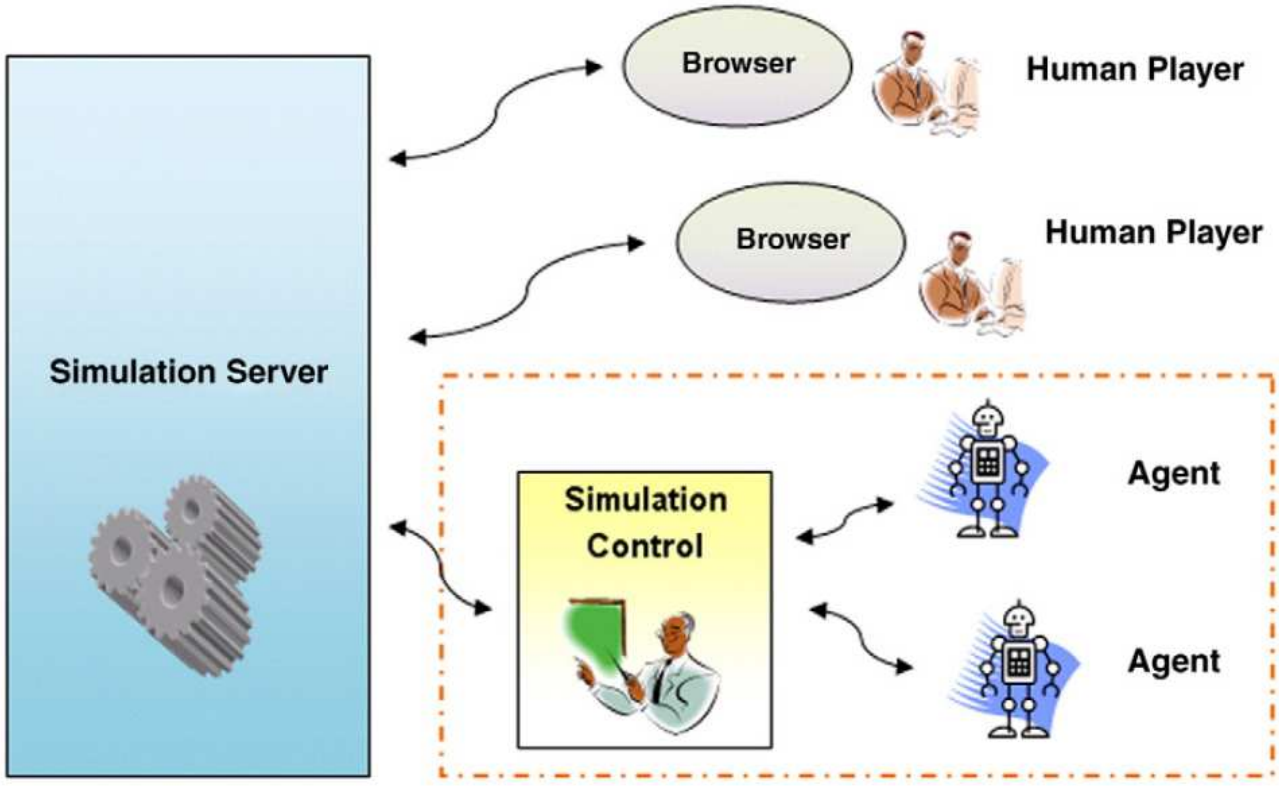


Figura 4: Arquitectura Software de SIMBA como sistema multiagente

El sistema está formado por los siguientes componentes:

-**Servidor de simulación (Simulation Server)**: recibe las decisiones de los participantes siguiendo el protocolo SOAP (Simple Object Access Protocol). Además, una vez que todas las decisiones son tomadas en un arbitraje, el motor de razonamiento del simulador calcula las variables de mercado para cada participante. Después del arbitraje, el servidor de simulación envía los resultados calculados a los participantes, que serán usados para volver a hacer la toma de decisiones del siguiente periodo de simulación.

-**Control de simulación (Simulation Control)**: Este componente es muy importante en este proyecto porque es la base de un programa llamado *Experimenter*. Este programa es utilizado para realizar pruebas en serie, agilizando los trámites de evaluación y prueba del motor de razonamiento basado en reglas. *Experimenter* gestiona los agentes software y sus decisiones. El control de simulación recibe las decisiones tomadas por los agentes software y las manda al motor de razonamiento de la simulación. El motor de simulación calcula los resultados de cada agente software y las manda de vuelta a *Experimenter*. Finalmente *Experimenter* manda sus resultados al agente software correspondiente para volver a repetir este ciclo de ejecución de arbitrajes en serie. Podemos concluir que *Experimenter* permite realizar de manera automática una gran cantidad de arbitrajes seguidos. Para realizar el control de la simulación se ha usando el lenguaje de programación Java y Sockets seguros.

-**Agente software (Agent)**: representa la alternativa al participante humano. En cada arbitraje, los agentes software reciben el resultado calculado por el motor de razonamiento de la simulación. El agente software usa esta información para tomar las decisiones del siguiente arbitraje, de igual manera que lo haría un participante humano.

En la actualidad existen tres tipos de agentes software como alternativas a los agentes humanos. Todas las acciones que pueden desarrollar estos agentes están limitadas por la semántica del modelo empresarial, así que se asume que los agentes solo toman decisiones que están dentro del rango. Estos tres tipos de agentes son:

-Agentes aleatorios: eligen la mejor toma de decisiones utilizando números aleatorios uniformes. También se usan agentes de este tipo para la investigación espacial.

-Agentes codificados manualmente: establece las variables de decisión utilizando una diferencia (positiva o negativa) definida por programador. Este comportamiento es más inteligente que el seguido por los agentes aleatorios.

-Agentes de aprendizaje por refuerzo: estos agentes usan el estado actual del entorno para elegir la mejor toma de decisiones en cada periodo de decisión.

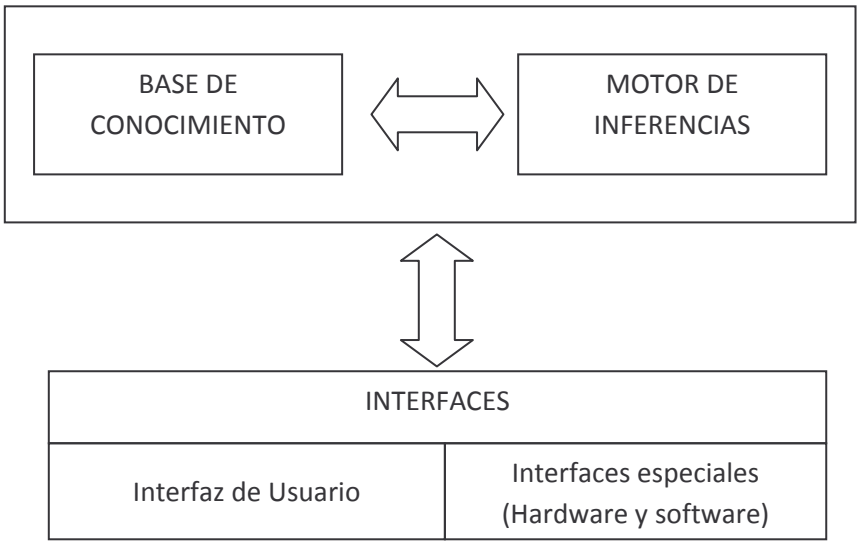
2.5. Sistemas basados en el conocimiento

Los sistemas basados en el conocimiento (SBC), se definen como el conjunto de principios, métodos y herramientas que permiten aplicar el saber científico y de la experiencia a la utilización de los conocimientos y de sus fuentes, mediante invenciones o construcciones útiles para el hombre [1].

Un SBC se compone de una estructura de datos y relaciones de carácter declarativo y procedimental junto a representaciones heurísticas que forman el conocimiento de un sistema [2]. Además, poseen métodos para realizar inferencias sobre el conocimiento almacenado, procesos más o menos complejos de búsqueda, toma de decisiones en problemas complejos, y a menudo incorporan mecanismos de aprendizaje y comunicación en lenguaje natural.

Los SBC se aplican a dominios y problemas más complejos de los que trata la informática tradicional. Las salidas que proporcionan los SBC necesitan de procesos elaborados, como métodos para deducir nueva información y técnicas heurísticas para reducir los espacios de búsqueda de la solución. Los SBC también son declarativos y heurísticos, utilizan bases de conocimiento y métodos de resolución de problemas que se adaptan al estado del problema, sobre los que se pueden modificar los objetivos de la solución, y que pueden ir optimizando su comportamiento a lo largo del tiempo.

Con respecto a la arquitectura de los SBC, se puede decir que tienen dos partes principales: la base de conocimiento, y el motor de inferencia o proceso de razonamiento. Sin embargo los SBC también constan de otras partes como son la base de hechos, la interfaz y el módulo de explicación.



Aunque los componentes principales de un SBC son la base de conocimiento y el motor de inferencia, también existen otras características importantes como la capacidad para el tratamiento de la incertidumbre o para generar explicaciones de su línea de razonamiento.

Por tanto la principal característica de un SBC es el potente cuerpo de conocimientos que acumula durante la construcción del sistema [3]. Estos conocimientos son explícitos y están organizados para simplificar la toma de decisiones. De hecho, la acumulación y codificación de conocimientos es uno de los aspectos más importantes de un SBC.

La información procesada que forma la base de conocimiento, debe ser por una parte de naturaleza declarativa (a menudo de definición técnica o de diccionario), y por otra parte debe ser prejuiciosa, para que pueda ser reutilizada en otros sistemas. De esta forma, se entiende que los conocimientos se almacenan en forma de:

- Definiciones descriptivas de términos específicos del dominio.
- Descripciones de objetos individuales del dominio, y sus relaciones con otros objetos.
- Criterios para tomar decisiones

la base de conocimiento también tiene que tener algunos tipos de conocimientos procedimentales tales como descripciones de comportamientos o procesos, que constituyen, junto con los conocimientos englobados en el motor de inferencias, el "*know how*" del SBC.

Los SBC difieren claramente del resto de aplicaciones que se producen en IA en varios aspectos. En primer lugar, ejecutan tareas difíciles con las prestaciones de un experto, En segundo término, enfatizan estrategias de solución de problemas de dominios específicos más que métodos generales, pero ineficientes de la IA. En tercer lugar, emplean autoconocimiento para razonar acerca de sus propios procesos de inferencia y proporcionan explicaciones o justificaciones para las conclusiones obtenidas.

Por tanto, un SBC se puede describir utilizando tres niveles diferentes:

Funcional: En este nivel, se hace una descripción funcional de cómo aparece el sistema ante el usuario. Es decir, se muestra la arquitectura del sistema que es aquello que ve y conoce el usuario.

Lógico: Que es el que se corresponde con la implementación que tiene que soportar la arquitectura ya descrita.

Físico: Que se corresponde con la realización concreta del sistema, esto es, con los componentes particulares empleados para construir el SBC.

2.6. CLIPS

Una manera de hacer operativos los Sistemas Basados en el Conocimiento es representarlos como sistemas de producción. Un sistema de producción proporciona una estructura que facilita la descripción y la ejecución de un proceso (en este caso de cálculo) para obtener unos resultados. Una herramienta de construcción de sistemas de producción consta de:

- Un conjunto de facilidades para la definición de reglas.
- Mecanismos para acceder a una o más bases de conocimiento y datos.
- Una estrategia de control que especifica el orden en el que las reglas son procesadas y la forma de resolver conflictos, que pueden aparecer cuando varias reglas coinciden simultáneamente.

Existen lenguajes especializados en sistemas de producción, encuadrados dentro de los que se conoce como lenguajes de Inteligencia Artificial. Uno de estos lenguajes es CLIPS

La herramienta de programación CLIPS (C Language Integrated Production System) fue desarrollada por la sección de inteligencia artificial de la NASA en los años ochenta, para solucionar algunos problemas que la agencia estaba encontrando en el desarrollo de sus sistemas expertos. En la actualidad, CLIPS es un lenguaje de programación de referencia en el área de los sistemas expertos, llegando incluso a desarrollarse conferencias sobre CLIPS y su aplicación.

CLIPS es una herramienta sencilla, pero que hace uso de un paradigma de programación llamado programación basada en reglas. La programación basada en reglas sigue una filosofía distinta a la programación procedimental o a la programación orientada a objetos, que son a las que la mayoría de los programadores están más habituados. Por ello es muy importante entender y adoptar los esquemas lógicos de programación basada en reglas para poder escribir un programa en CLIPS [6].

En este apartado vamos a explicar la manera en la que CLIPS estructura la información para generar una ontología, y posteriormente en función de esas estructuras de información, creamos reglas para definir el comportamiento que debe tomar el motor de razonamiento para generar los resultados de los cálculos que le indicaremos.

Inicialmente hay que explicar que CLIPS representa la información en objetos, que luego usará para hacerlos interactuar. Sin embargo no podemos crear objetos directamente, sino que debemos crear clases de las que luego crearemos objetos. Podríamos expresarlo mejor con un símil, donde la clase en la que vamos a representar la información sería el molde para dar forma a una galleta, y el objeto creado a partir de ese molde sería la galleta.

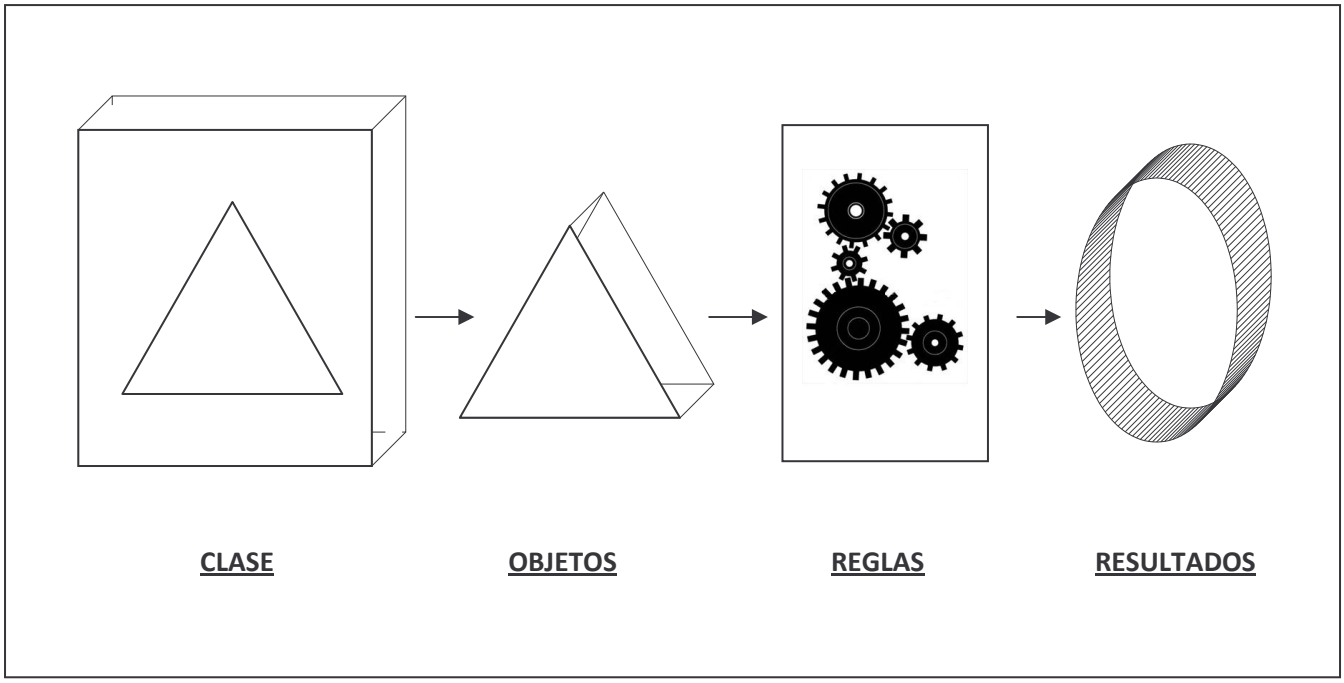


Figura 5: Esquema de obtención de resultados a través de la creación de objetos, clases y reglas

Los objetos con los que trabaja CLIPS pueden almacenarse en memoria de trabajo, y pueden ser de dos tipos:

- Plantillas: Registros compuestos por varios campos.
- Marcos: Clases similares a las de la filosofía de Programación Orientada a Objetos. Mediante una jerarquía de clases se representan desde el concepto más general (clase padre), hasta los más específicos (clases hijo).

Sin embargo en este proyecto se va a trabajar con marcos porque se adapta mejor al sistema de producción que queremos realizar para el motor de razonamiento basado en reglas.

Profundizando un poco más, podríamos decir que las clases están compuestas por atributos o slots, que almacenarán valores característicos de un objeto. Esto quiere decir que por ejemplo para la clase *dirección* podemos tener los atributos *tipo_via*, *nombre*, *numero*, *escalera*, *piso*, *letra*, *cod_postal*, *localidad* y *provincia*. Y se puede representar así:

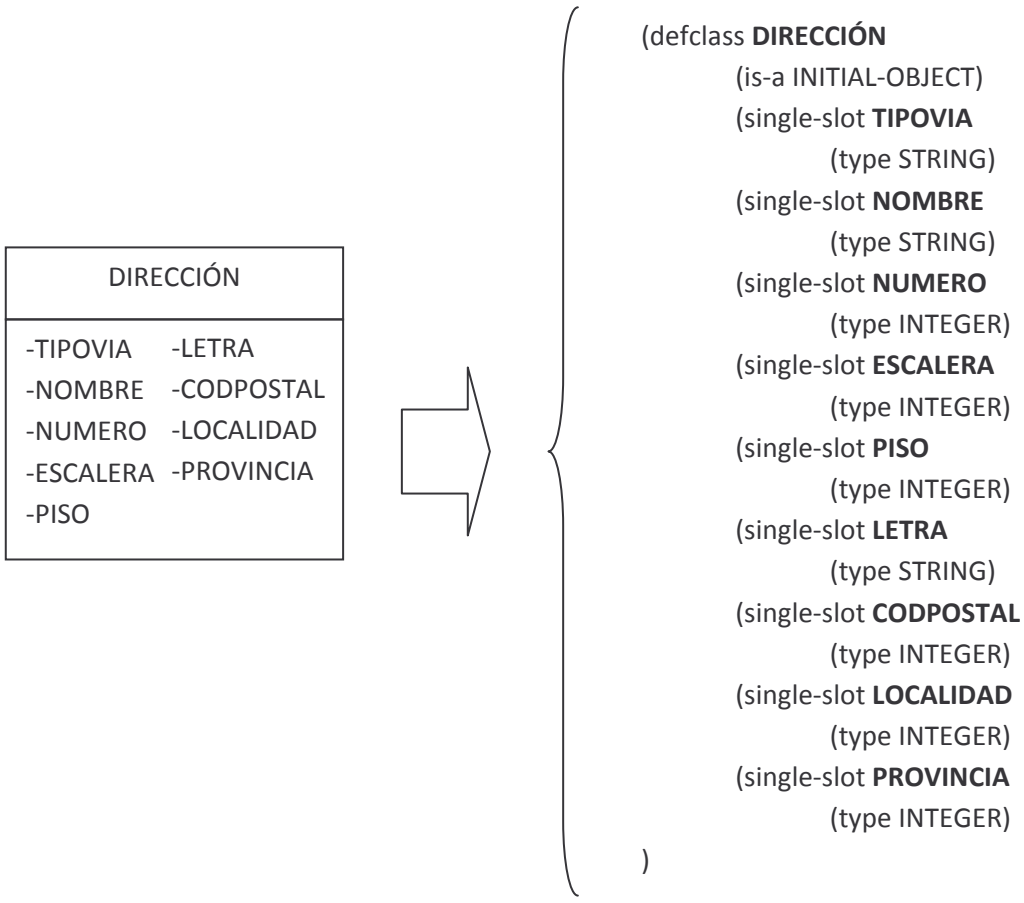
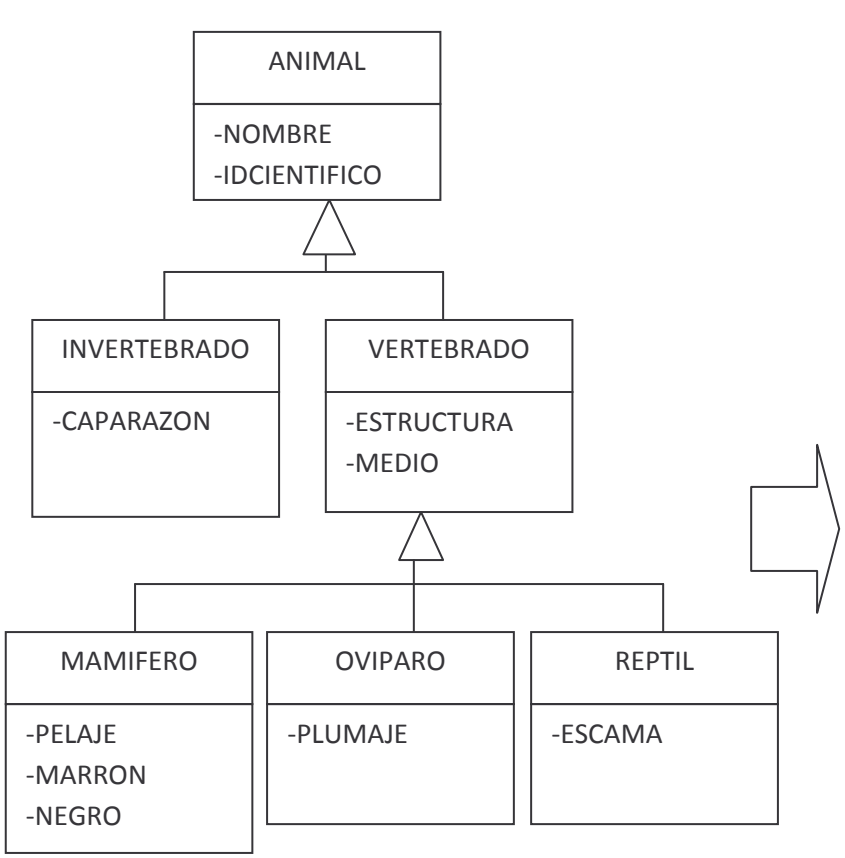


Figura 6: Ejemplo de representación de atributos de una clase

Como CLIPS es una herramienta de programación con una filosofía similar a la programación orientada a objetos, podemos aprovecharnos de algunas ventajas que este aspecto nos reporta, como la herencia entre clases. Esto quiere decir que se pueden crear clases que de un tipo ya definido en una clase padre superior. Además se pueden añadir atributos propios y característicos de la nueva clase que se desea crear. Un ejemplo de jerarquía con herencias es el siguiente:




```
(defclass ANIMAL
        (is-a INITIAL-OBJECT)
        (single-slot NOMBRE
                (type STRING)
        (single-slot IDCIENTIFICO
                (type INTEGER))
(defclass INVERTEBRADO
        (is-a ANIMAL)
        (single-slot CAPARAZON
                (type STRING))
(defclass VERTEBRADO
        (is-a ANIMAL)
        (single-slot ESTRUCTURA
                (type STRING)
        (single-slot MEDIO
                (type STRING))
(defclass MAMIFERO
        (is-a VERTEBRADO)
        (single-slot PELAJE
                (type STRING)
        (single-slot MARRON
                (type STRING)
(defclass OVIPARO
        (is-a VERTEBRADO)
        (single-slot PLUMAJE
                (type STRING))
(defclass REPTIL
        (is-a VERTEBRADO)
        (single-slot ESCAMA
                (type STRING))
```

Figura 7: Ejemplo de representación jerarquía entre clases

Lo que se ha explicado hasta este punto son las estructuras y jerarquías en las que se puede descomponer la información, es decir las clases. El conjunto de estas clases y sus jerarquías forman la ontología genérica del sistema de producción. Ahora se verá cómo se gestiona la información para obtener resultados.

Para definir el comportamiento que un sistema de producción debe seguir (operaciones, cálculos matemático, etc.), se utiliza una base de reglas. La base de reglas está compuesta por reglas.

Las reglas tienen dos partes:

- La parte izquierda, donde se encuentran las condiciones necesarias para que se dispare la regla. Hay que reseñar que deben cumplirse todas y cada una de las condiciones para que se active la regla.

- La parte derecha, donde se describen las acciones que se realizarán cada vez que se dispare la ejecución de la regla.

Según esto, todas las reglas de la base de reglas de un sistema de producción tendrían la siguiente estructura:

```
(defrule COMPROBAR_PELO_MARRON
    ?animal <- (object (is-a MAMIFERO) (PELAJE ?pelaje))
    (test (= ?pelaje "MARRON"))
    =>
    (modify-instance ?animal (MARRON "SI"))
```

PARTE IZQUIERDA

PARTE DERECHA

Como ya se ha explicado, CLIPS no sigue un hilo de ejecución definido, sino que actúa en función de las reglas que pueda ejecutar en cada momento. Esto quiere decir que para saber qué regla debe ejecutarse en cada momento, el motor de inferencias tiene que elegir entre un conjunto de reglas posibles a ejecutar.

Para resolver este conflicto CLIPS realiza una serie cíclica de tres fases. En cada una de ellas se realiza lo siguiente:

- Equiparación: se activan las reglas cuyas partes izquierdas equiparan con las instancias existentes.
- Resolución del conjunto conflicto: se selecciona una de las reglas activadas en la fase anterior. La elección se realiza según los criterios indicados por la estrategia de control.
- Ejecución: se realizan las acciones indicadas en la parte derecha de la regla seleccionada.

Un sistema de producción codificado en CLIPS permite recibir entradas, que en el caso del motor de razonamiento que se ha implementado es en forma de ficheros de entrada. Este fichero contiene datos referentes al historial y a las empresas implicadas.

De la misma manera, el motor de razonamiento almacena los resultados obtenidos en un arbitraje en un fichero de salida.

Capítulo 3

Objetivos del proyecto

El objetivo general de este proyecto es desarrollar un nuevo motor de razonamiento basado en reglas para el simulador empresarial SIMBA.

Los objetivos específicos del proyecto son:

- Estudio y análisis crítico del simulador: El primer objetivo específico es analizar el motor de razonamiento basado en programación estructural que ya se encontraba implementado para el proyecto SIMBA. El análisis inicial permitirá esquematizar el esqueleto del motor de razonamiento y ver en que sectores diferentes se divide. También servirá para entender la interacción que hay entre el equipo de toma de decisiones y el posterior arbitraje de mercado en un periodo. A partir de este análisis inicial se va a construir la estructura de datos para representar la información.

- Familiarización con la tecnología: Otro objetivo específico es la familiarización profunda con el entorno UNIX, la programación orientada a objetos, los sistemas basados en reglas, la programación en c++, la herramienta CLIPS y la gestión dinámica de Bases de Datos mediante MySQL en red.

- Implementación del motor de razonamiento: Sin duda el objetivo específico con mayor relevancia es la implementación del motor de razonamiento basado en reglas. Es por ello que engloba varios puntos clave en su desarrollo e implementación:

    - Diseño de la ontología: El primer paso en la implementación del motor de razonamiento es definir una estructura eficiente, sólida y robusta de datos que se ajuste a las necesidades que encontraremos en una simulación de mercado.

    - Análisis de reglas: Una vez tenemos la ontología el siguiente objetivo es realizar un análisis crítico y profundo de las reglas que posteriormente se van a implementar en CLIPS.

- Diseño del motor: Se va a realizar un diseño eficiente de las reglas por las que se regirán los diferentes módulos de nuestro motor (calidad, tesorería, etc.). Estos comportamientos se basarán en algoritmos matemáticos de mercado facilitados por la empresa Simuladores Empresariales S.L. con gran experiencia en el campo.

- Implementación del motor basado en reglas: A su vez los algoritmos serán traducidos a CLIPS para realizar un conjunto de reglas optimizado de comportamiento de mercado.

- Acceso a la BBDD: Debido al continuo cambio de los datos almacenados en la Base de Datos de este proyecto, un objetivo específico es la comunicación directa con un servidor Web mediante MySQL. Esto nos permitirá realizar en cada momento arbitrajes de mercado exactos y ajustados a la realidad *just-on-time* de las empresas.

- Gestión de resultados: Otro objetivo específico es gestionar las predicciones de mercado que se generarán y guardarán en un historial ordenadamente para su posterior consulta en el servidor Web.

- Integración del motor en SIMBA: El objetivo final de la implementación será la integración del nuevo motor de razonamiento basado en reglas en el proyecto SIMBA, sustituyéndolo por el anterior motor estructural.

- Validación y pruebas: Un punto muy importante de este proyecto es realizar pruebas del motor de razonamiento basado en reglas para comprobar que produce resultados coherentes dentro de un rango de valores en el ámbito económico.

- Evaluación de tiempos de ejecución: Después de realizar la depuración lógica (que todo funcione correctamente) realizada en el punto anterior, el siguiente objetivo es realizar una depuración de eficiencia (que hace todo correctamente con los mínimos recursos posibles, en este caso, el recurso es el tiempo). De esta manera se pretende evaluar los tiempos de ejecución obtenidos con el nuevo motor de razonamiento en comparación con los tiempos de ejecución obtenidos con el antiguo motor de SIMBA.

Capítulo 4

Memoria de trabajo

En este capítulo se explican detalladamente las consideraciones iniciales y los pasos seguidos para el desarrollo del motor de razonamiento basado en reglas. En el apartado 4.1 se ofrece una visión general de la arquitectura. En el apartado 4.2 se analizan los elementos fundamentales de la base del conocimiento del sistema. En el apartado 4.3 se describe la creación de la base de reglas que determina el comportamiento del motor de razonamiento. Finalmente en el apartado 4.4 se explica la integración de motor de razonamiento basado en reglas en el simulador SIMBA.

En algunos de los apartados de este capítulo se ha omitido información sensible (formulas, algoritmos, etc.) para cumplir el acuerdo de confidencialidad. La información omitida está disponible en el anexo I, que solamente se entregará al tribunal evaluador para el correcto entendimiento de este proyecto.

4.1. Arquitectura de la aplicación

El diseño general de la arquitectura del sistema abarca más procesos que el motor de razonamiento basado en reglas. Esto quiere decir que para realizar satisfactoriamente un arbitraje, es necesario que intervengan más procesos (comunicación, entradas, salidas, etc.), interactuando con el motor de razonamiento basado en reglas. La arquitectura de la aplicación está formada por los siguientes componentes:

- **Bases de datos**: En este proyecto se trabaja con una base de datos SQL alojada en el servidor del simulador SIMBA, que contiene toda la información referente a las empresas y su historial, los diferentes entornos económicos, etc. En el inicio y en el final de un arbitraje, es necesaria una comunicación fluida entre la base de datos y el motor de razonamiento basado en reglas.

- **Módulo de generación de Historial**: Este módulo se encarga de obtener los datos necesarios de la base de datos para generar un historial en formato CLIPS, y enviárselo al motor de razonamiento basado en reglas.

- **Motor de razonamiento basado en reglas**: El motor de razonamiento basado en reglas es el que se encarga de realizar el proceso de arbitraje de cada empresa en cada periodo. Como se explico en el apartado 2.3 el proceso de arbitraje está determinado por un entorno económico y un mercado.

- **Módulo de procesado de resultados**: Una vez que se ha terminado el proceso de arbitraje en el motor de razonamiento basado en reglas, este módulo se encarga de introducir en la base de datos alojada en el servidor los resultados generados para un periodo.

La arquitectura global de la aplicación se representa en la figura 8:

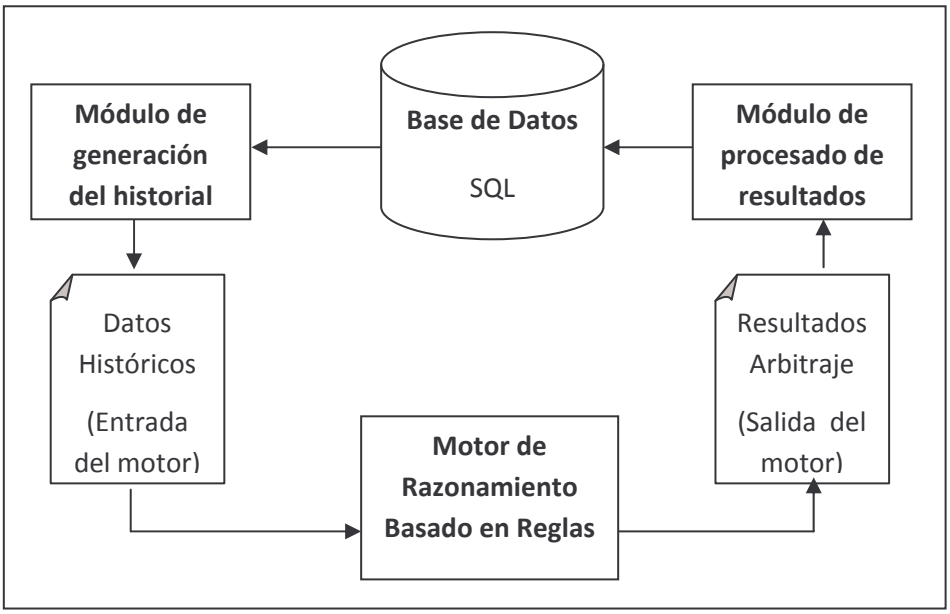


Figura 8: Arquitectura global de la aplicación

La dinámica de la ejecución de la aplicación comienza cuando el módulo de generación de historial crea un fichero dinámico, sacando los datos del histórico de la base de datos alojada en el servidor. Este fichero contiene todos los datos históricos que son necesarios para arbitrar un periodo (Historial Decisiones, Historial AreaComercial, etc). El fichero creado se envía al motor de razonamiento basado en reglas, que realiza el proceso de arbitraje en función de los datos de entrada recibidos en el fichero.

Después del proceso de arbitraje, se generan unos resultados en el fichero de salida del motor de razonamiento basado en reglas, que se envía al módulo de procesado de resultados. En este módulo se analizan los resultados generados, y se introducen en la base de datos del servidor.

Este proceso se repite cada vez que se produce un arbitraje en el simulador SIMBA.

4.2. Modelo de conocimiento

La información está estructurada siguiendo el diagrama entidad relación (E/R) de la figura 9 del Anexo I, donde podemos ver las entidades que componen el sistema SIMBA, y la comunicación que mantienen. En función de esta información, se han esquematizado las estructuras o clases que utilizaremos en el motor de razonamiento para facilitar los cálculos de los algoritmos con el fin de realizar una predicción de mercado eficaz y sobre todo eficiente en cuanto a la gestión de información. Según esto, se han creado clases para tres ámbitos diferentes: Historial, Constantes y Resultados como se indica en el siguiente esquema:

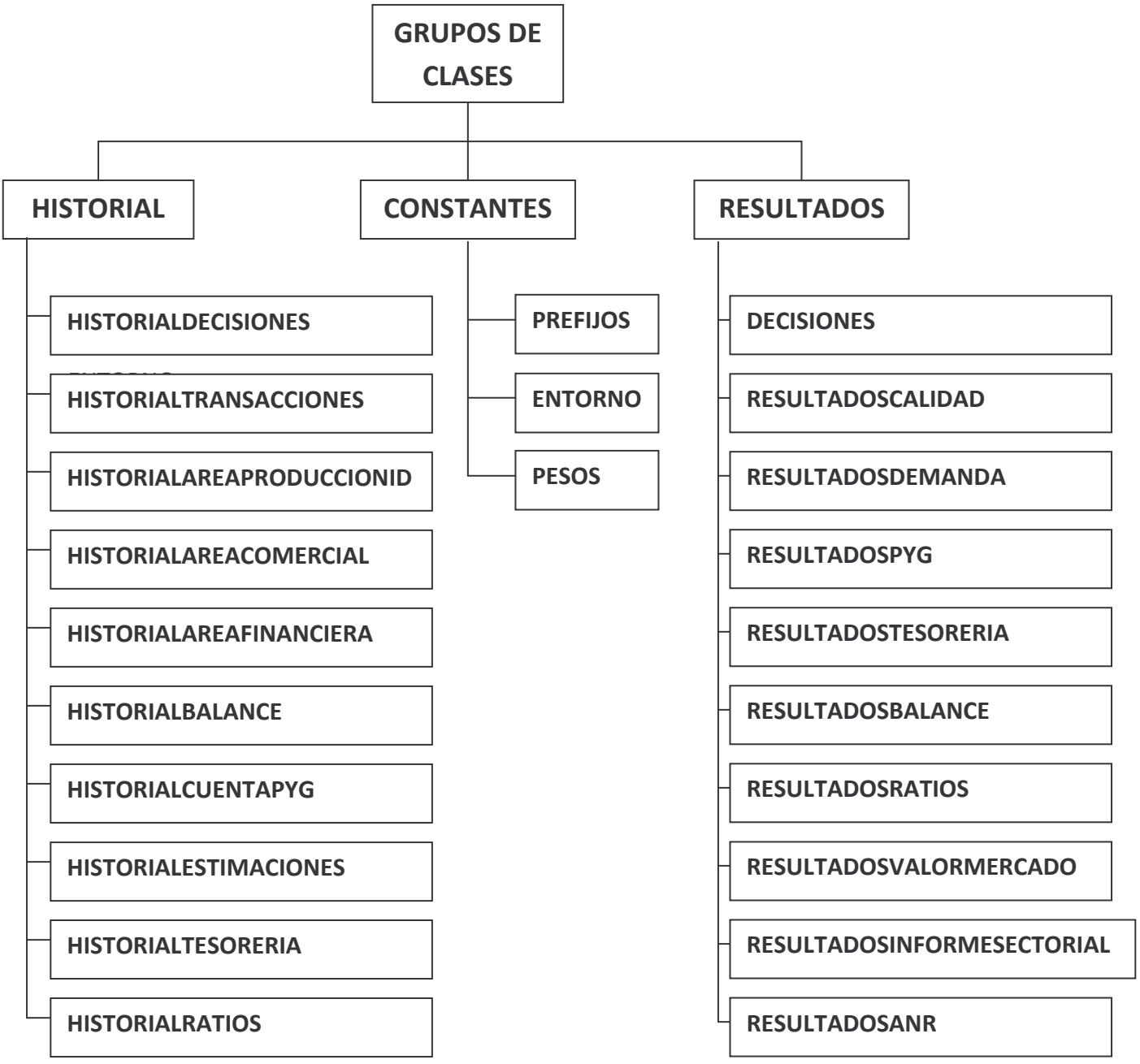


Figura 10: Jerarquía de clases del motor de razonamiento basado en reglas

Ahora vamos a presentar las clases que se han creado para el motor de razonamiento basado en reglas comenzando por el ámbito del historial. Posteriormente las clases creadas para representar la información del ámbito de las constantes, y finalmente las clases creadas para almacenar los resultados de todos los cálculos del motor de razonamiento.

Historial

El historial es el informe donde se almacenan todas las variables que tiene una empresa en diversos campos como por ejemplo las decisiones que ha tomado, las transacciones que ha tenido, su área financiera etc. Estas variables estarán almacenadas en la base de datos del servidor, para posteriormente ser consultadas por el motor de razonamiento. A partir de la información recibida, el motor de razonamiento realizará unos cálculos para determinar una estimación de los valores en el futuro, en función de unas decisiones tomadas. Para mayor eficiencia se ha generado una clase por cada uno de los diferentes ámbitos en los que se dividen las variables del historial de una empresa, que son las siguientes:

- **HISTORIALDECISIONES**: En esta clase se almacenan todas las variables de la empresa referentes a las decisiones que el directivo debe tomar para determinar el rumbo que tomará la empresa en el futuro. Se puede encontrar un desglose pormenorizado de los atributos que componen esta clase y del resto de clases que componen el historial dentro del ***ANEXO I***. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALDECISIONES	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
DPRECIOEMP	*FLOAT*
DPUBLICIDA	*FLOAT*
[…]	
DDIVRES	*FLOAT*

Figura 11: Resumen de la clase *HISTORIALDECISIONES*

Y se ha implementado así:

```
(defclass HISTORIALDECISIONES
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot DPRECIOEMP
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
    (single-slot DPUBLICIDA
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot DDIVRES
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **HISTORIALTRANSACCIONES**: En esta clase se almacenan todas las variables de la empresa referentes a las transacciones que se han realizado con otras empresas tanto de compra como de venta. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALTRANSACCIONES	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*STRING*
PERIODO	*INTEGER*
DEMPCOM	*FLOAT*
[…]	
DVTASCTDO	*FLOAT*

Figura 12: Resumen de la clase *HISTORIALTRANSACCIONES*

Y se ha implementado así:

```
(defclass HISTORIALTRANSACCIONES
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot DEMPCOM
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot DVTASCTDO
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **HISTORIALAREAPRODUCCIONID**: En esta clase se almacenan todas las variables de la empresa referentes a la producción del producto, el stock producido, los recursos de plantilla, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALAREAPRODUCCIONID	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
RCAPINST	*FLOAT*
ROCUP	*FLOAT*
[…]	
RCOBREDCAP	*FLOAT*

Figura 13: Resumen de la clase *HISTORIALAREAPRODUCCIONID*

Y se ha implementado así:

```
(defclass HISTORIALAREAPRODUCCIONID
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot RCAPINST
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
    (single-slot ROCUP
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot RCOBREDCAP
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **HISTORIALAREACOMERCIAL**: En esta clase se almacenan todas las variables de la empresa referentes al entorno de comercio como la calidad, el stock de ventas, el coste de fabricación, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

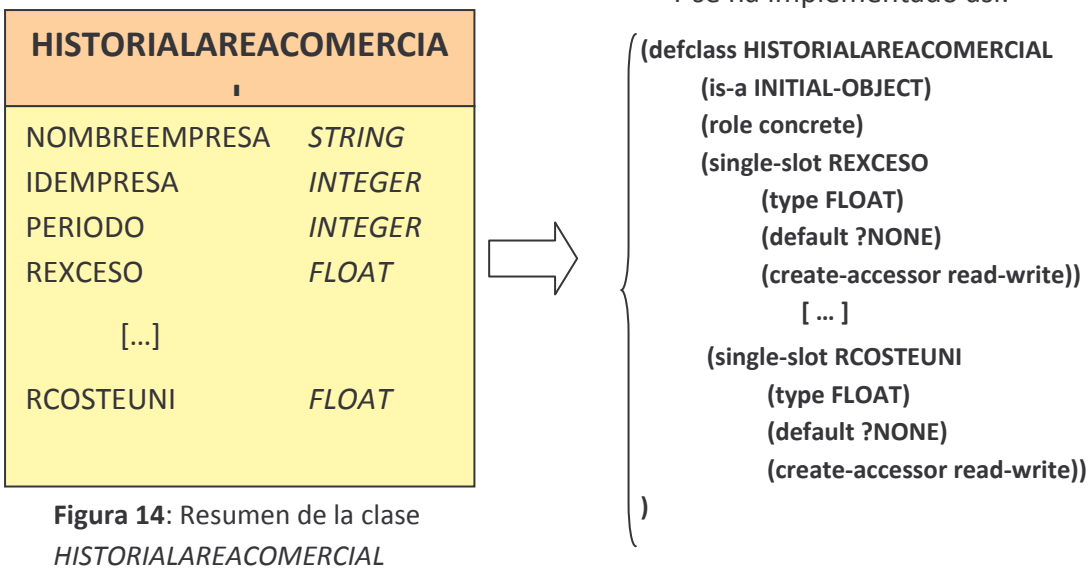


Figura 14: Resumen de la clase *HISTORIALAREACOMERCIAL*

- **HISTORIALAREAFINANCIERA**: En esta clase se almacenan todas las variables de la empresa referentes al ámbito financiero como por ejemplo los dividendos, el valor contable, el valor de mercado, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

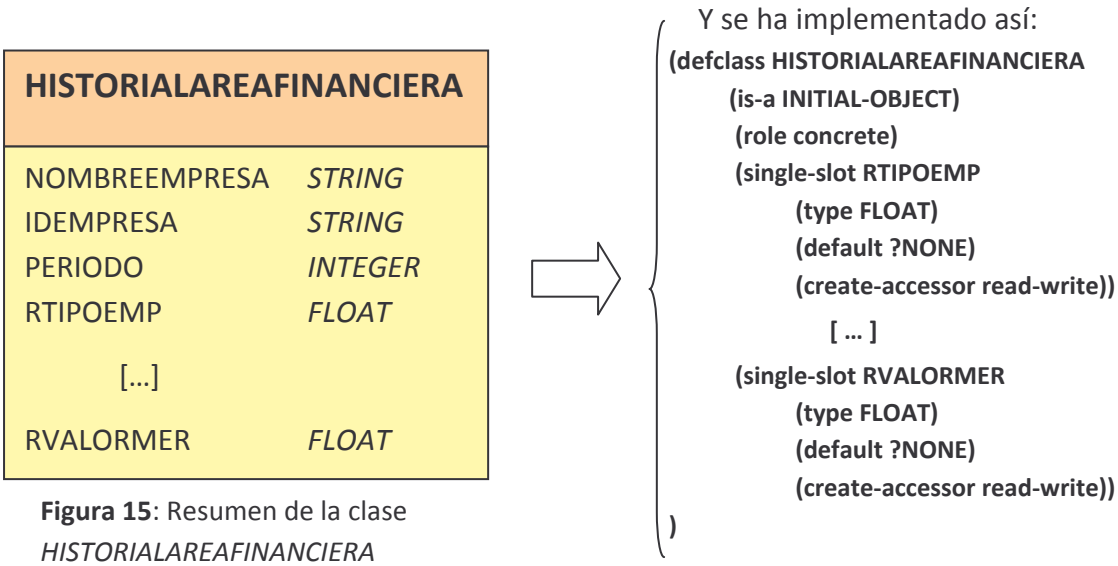


Figura 15: Resumen de la clase *HISTORIALAREAFINANCIERA*

- **HISTORIALBALANCE**: En esta clase se almacenan todas las variables de la empresa referentes al balance financiero como los gastos de amortización, los gastos de establecimiento, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALBALANCE	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
RGTOSESTAB	*FLOAT*
RGTOSAMOR	*FLOAT*
[...]	
RACUMPYG	*FLOAT*

Figura 16: Resumen de la clase *HISTORIALBALANCE*

Y se ha implementado así:

```
(defclass HISTORIALBALANCE
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot RGTOSESTAB
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
      (single-slot RGTOSAMOR
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
            [ … ]
      (single-slot RACUMPYG
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **HISTORIALCUENTAPYG**: En esta clase se almacenan todas las variables de la empresa referentes a la cuenta de pérdidas y ganancias. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALCUENTAPYG	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
RVTANETA	*FLOAT*
[...]	
RGTOSCOM	*FLOAT*

Figura 17: Resumen de la clase *HISTORIALCUENTAPYG*

Y se ha implementado así:

```
(defclass HISTORIALCUENTAPYG
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot RVTANETA
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
            [ … ]
      (single-slot RGTOSCOM
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **HISTORIALESTIMACIONES**: En esta clase se almacenan todas las variables de la empresa referentes a las estimaciones que se realizan como por ejemplo la demanda estimada, el precio de publicidad estimado, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALESTIMACIONES	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*STRING*
PERIODO	*INTEGER*
RAGREGAEST	*FLOAT*
[...]	
ESTACCVEN	*FLOAT*

Figura 18: Resumen de la clase *HISTORIALESTIMACIONES*

Y se ha implementado así:

```
(defclass HISTORIALESTIMACIONES
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot RAGREGAEST
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot ESTACCVEN
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **HISTORIALTESORERIA**: En esta clase se almacenan todas las variables referentes a la tesorería de cada empresa como por ejemplo los ingresos por ventas, el valor contable de las acciones, el saldo de la tesorería, etc. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALTESORERIA	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*STRING*
PERIODO	*INTEGER*
RVTASPLA	*FLOAT*
[...]	
RTESORE	*FLOAT*

Figura 19: Resumen de la clase *HISTORIALTESORERIA*

Y se ha implementado así:

```
(defclass HISTORIALTESORERIA
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot RVTASPLA
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot RTESORE
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **HISTORIALRATIOS**: En esta clase se almacenan todas las variables de la empresa referentes a los ratios usados como por ejemplo el ratio de liquidez o el ratio de cotización entre otros. Hemos resumido todas las variables y su tipo en la siguiente estructura:

HISTORIALRATIOS	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
RLIQUI	*FLOAT*
[…]	
RPER	*FLOAT*

Figura 20: Resumen de la clase *HISTORIALRATIOS*

Y se ha implementado así:

```
(defclass HISTORIALRATIOS
        (is-a INITIAL-OBJECT)
        (role concrete)
        (single-slot RLIQUI
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
                    [ … ]
        (single-slot RGTOSCOM
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
)
```

Constantes

Las constantes, multiplicadores y cuantificadores que se utilizan en los cálculos del motor de razonamiento basado en reglas definen un entorno común para todas las empresas de la simulación. Con todas estas constantes se definen ámbitos como el precio de la materia prima, un cuantificador de demanda, el tipo de interés o la tasa de amortización inmovilizada entre otros. Estos cuantificadores están divididos en tres grupos que son Entorno, Prefijos y Pesos:

- **ENTORNO**: En esta clase se almacenan todas las variables que tienen que ver con el entorno que rodean a todas las empresas, como el IPC o la tasa de crecimiento de ventas entre otras. Se puede encontrar un desglose pormenorizado de los atributos que componen esta clase, y del resto de clases que componen las constantes en el ***ANEXO I***. Hemos resumido todas las variables y su tipo en la siguiente estructura:

ENTORNO	
PERIODO	*INTEGER*
PIPC	*FLOAT*
[…]	
PVALNOM	*FLOAT*

Figura 21: Resumen de la clase *ENTORNO*

Y se ha implementado así:

```
(defclass ENTORNO
        (is-a INITIAL-OBJECT)
        (role concrete)
        (single-slot PIPC
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
                    [ … ]
        (single-slot PVALNOM
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
)
```

- **PREFIJOS**: En esta clase se almacenan todas las variables que tienen que ver con multiplicadores que intervienen en los algoritmos de cálculo de diferentes módulos como el de calidad. Hemos resumido todas las variables y su tipo en la siguiente estructura:

PREFIJOS	
PPCALIDADT-1	*FLOAT*
[…]	
PPCALIDADRED	*FLOAT*

Figura 22: Resumen de la clase *PREFIJOS*

Y se ha implementado así:

```
(defclass PREFIJOS
        (is-a INITIAL-OBJECT)
        (role concrete)
        (single-slot PPCALIDADT_1
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
                    [ … ]
        (single-slot PPCALIDADRED
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
)
```

- **PESOS**: En esta clase se almacenan todas las variables que tienen que ver con los cuantificadores que se usan en los algoritmos de cálculo de diferentes módulos. Hemos resumido todas las variables y su tipo en la siguiente estructura:

PESOS	
PPPTOSCALIDADI+DT-1A	*FLOAT*
PPPTOSCALIDADI+DT-1B	*FLOAT*
PPPTODCALIDADI+DT-2A	*FLOAT*
PPPTODCALIDADI+DT-2B	*FLOAT*
PPPTOSCALIDADI+DT-3A	*FLOAT*
PPPTOSCALIDADI+DT-3B	*FLOAT*
PPPTOSCALIDADI+DSEGT-1A	*FLOAT*
PPPTOSCALIDADI+DSEGT-1B	*FLOAT*
PPPTOSCALIDADI+DSEGT-2A	*FLOAT*
PPPTOSCALIDADI+DSEGT-2B	*FLOAT*
PPPTOSCALIDADI+DSEGT-3A	*FLOAT*
PPPTOSCALIDADI+DSEGT-3B	*FLOAT*
[…]	
PPGGFORMLIM2	*FLOAT*

Figura 24: Resumen de la clase *PESOS*

Y se ha implementado así:

```
(defclass PESOS
        (is-a INITIAL-OBJECT)
        (role concrete)
        (single-slot PPPTOSCALIDADI+DT-1A
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
        (single-slot PPPTOSCALIDADI+DT-1B
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))

                    [ … ]

        (single-slot PPGGFORMLIM2PVALNOM
                (type FLOAT)
                (default ?NONE)
                (create-accessor read-write))
)
```

Resultados

Los resultados obtenidos en los cálculos determinarán el comportamiento que seguirá una empresa, en un entorno determinado y para unas decisiones tomadas. Por lo tanto estos resultados son muy importantes porque a partir de ellos se escogerá el camino a seguir en función de la obtención siempre del máximo beneficio o menores pérdidas. Estos resultados se almacenarán en la base de datos del servidor, de la misma manera que estaba constituido el historial desde el que partíamos para iniciar un arbitraje. Por lo tanto, después de un arbitraje, tenemos que enviar estos resultados de vuelta al servidor.

Es importante destacar que estos resultados obtenidos para un periodo van a definir casi completamente el historial del siguiente periodo. De manera que una vez arbitrado un periodo, inmediatamente se va a convertir en el historial del arbitraje del siguiente periodo, junto con unas nuevas decisiones que se tomen para determinar el futuro rumbo financiero a seguir.

Para realizar una gestión de datos más eficiente, se ha dividido toda la información de los resultados en función del módulo en el que se calculan. De esta manera se han creado las siguientes clases:

- **DECISIONES**: En esta clase se almacenan decisiones que se han obtenido de los resultados del cálculo del módulo de calidad. Estas decisiones intervendrán en el cálculo final de la variable RCALI (*Cal014*). Se puede encontrar un desglose pormenorizado de los atributos que componen esta clase y del resto de clases que componen los resultados en el ***ANEXO I***. Hemos resumido todas las variables y su tipo en la siguiente estructura:

DECISIONES	
DECI003	*FLOAT*
DECI004	*FLOAT*
[…]	
DECI019	*FLOAT*
DECI020	*FLOAT*

Figura 25: Resumen de la clase *DECISIONES*

Nota: Las decisiones DECI001 y DECI002 ya venían determinadas, por lo que no es necesario que sean almacenadas.

Y se ha implementado así:

```
(defclass DECISIONES
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot DECI003
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
            [ … ]
      (single-slot DECI020
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **RESULTADOSCALIADD**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de calidad. Este módulo es diferente al resto en el sentido de que todos los cálculos son intermedios excepto el *CAL014* que sirve para calcular el índice de calidad percibida (*RCALI*), lo que conlleva que en este modulo se realicen muchos cálculos preparatorios para facilitar y hacer más eficientes otros cálculos posteriores, como por ejemplo la ejecución de los sumatorio de sumatorio (PE: $\sum \sum DID$). Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSCALIDAD	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*INTEGER*
PERIODO	*INTEGER*
ESTADOCAL	*FLOAT*
CAL001A1	*FLOAT*
CAL001B1	*FLOAT*
[...]	
CALEST	*FLOAT*

Figura 26: Resumen de la clase *RESULTADOSCALIDAD*

Y se ha implementado así:

```
(defclass RESULTADOSCALIDAD
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot ESTADOCAL
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))

            [ … ]
      (single-slot CAEST
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **RESULTADOSDEMANDA**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de demanda. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSDEMANDA	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADODEM	*FLOAT*
ESTADODEM	*FLOAT*
DEM001	*FLOAT*
DEM002W	*FLOAT*
DEM002WCONT	*FLOAT*
[...]	
DEM023	*FLOAT*

Figura 27: Resumen de la clase *RESULTADOSDEMANDA*

Y se ha implementado así:

```
(defclass RESULTADOSDEMANDA
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot DEM001
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))

            [ … ]
      (single-slot DEM023
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **RESULTADOSPYG**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de pérdidas y ganancias. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSPYG	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOPYG	*FLOAT*
PYG001	*FLOAT*
PYG002A	*FLOAT*
PYG002B	*FLOAT*
PYG002C	*FLOAT*
[...]	
PYG027	*FLOAT*

Y se ha implementado así:

```
(defclass RESULTADOSPYG
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot PYG001
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))

                  [ … ]
      (single-slot PYG027
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

Figura 28: Resumen de la clase *RESULTADOSPYG*

- **RESULTADOSTESORERIA**: En esta clase se almacenan los resultados de los cálculos del módulo de tesorería. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSTESORERIA	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOTES	*FLOAT*
TES028	*FLOAT*
TES029	*FLOAT*
[...]	
TES044C	*FLOAT*

Y se ha implementado así:

```
(defclass RESULTADOSTESORERIA
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot TES028
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))

                  [ … ]
      (single-slot TES044C
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

Figura 29: Resumen de la clase *RESULTADOSTESORERIA*

- **RESULTADOSBALANCE**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de pérdidas y ganancias. Hemos resumido las variables y su tipo en la siguiente estructura:

RESULTADOSBALANCE	
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOBAL	*FLOAT*
BAL045A	*FLOAT*
BAL045B	*FLOAT*
BAL046	*FLOAT*
[...]	
BAL072	*FLOAT*

Figura 30: Resumen de la clase *RESULTADOSBALANCE*

Y se ha implementado así:

```
(defclass RESULTADOSBALANCE
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot BAL045A
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))

        [ … ]
    (single-slot BAL072
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **RESULTADOSRATIOS**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de ratios. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSRATIOS	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADORAT	*FLOAT*
RAT073	*FLOAT*
RAT074	*FLOAT*
RAT075	*FLOAT*
[...]	
RAT087	*FLOAT*

Figura 31: Resumen de la clase *RESULTADOSRATIOS*

Y se ha implementado así:

```
(defclass RESULTADOSRATIOS
    (is-a INITIAL-OBJECT)
    (role concrete)
    (single-slot ESTADORAT
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
        [ … ]
    (single-slot RAT087
        (type FLOAT)
        (default ?NONE)
        (create-accessor read-write))
)
```

- **RESULTADOSVALORMERCADO**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de valor de mercado. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSVALORMERCA	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOVMC	*FLOAT*
VMC001	*FLOAT*
VMC002	*FLOAT*
VMC003	*FLOAT*
VMC004_A	*FLOAT*
VMC004_B	*FLOAT*
VMC004_C	*FLOAT*
[...]	
VMC007	*FLOAT*

Figura 32: Resumen de la clase *RESULTADOSVALORMERCADO*

Y se ha implementado así:

```
(defclass RESULTADOSVALORMERCADO
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot VMC001
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
      (single-slot VMC002
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))

            [ … ]
      (single-slot VMC007
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **RESULTADOSINFORMESECTORIAL**: En esta clase se almacenan los resultados de los cálculos del módulo de informe sectorial. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSINFORMESECTORIAL	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOISE	*FLOAT*
ISE001	*FLOAT*
ISE002	*FLOAT*
ISE003	*FLOAT*
ISE004	*FLOAT*
[...]	
ISE025	*FLOAT*

Figura 33: Resumen de la clase *RESULTADOSINFORMESECTORIAL*

Y se ha implementado así:

```
(defclass RESULTADOSINFORMESECTORIAL
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot ISE001
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
            [ … ]
      (single-slot ISE025
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

- **RESULTADOSANR**: En esta clase se almacenan los resultados de los cálculos realizados en el módulo de análisis decisiones y ranking. Hemos resumido todas las variables y su tipo en la siguiente estructura:

RESULTADOSANR	
NOMBREEMPRESA	*STRING*
IDEMPRESA	*NTEGER*
PERIODO	*INTEGER*
ESTADOANR	*FLOAT*
ANR000AX	*FLOAT*
ANR000AY	*FLOAT*
ANR000AZ	*FLOAT*
ANR000BX	*FLOAT*
[...]	
ANR038	*FLOAT*

Figura 34: Resumen de la clase *RESULTADOSANR*

Y se ha implementado así:

```
(defclass RESULTADOSRATIOS
      (is-a INITIAL-OBJECT)
      (role concrete)
      (single-slot ANR000AX
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
                [ … ]
      (single-slot ANR038
            (type FLOAT)
            (default ?NONE)
            (create-accessor read-write))
)
```

4.3. Base de reglas

En este apartado se van a describir las reglas del motor de razonamiento que se han creado para realizar la funcionalidad necesaria del proyecto. Estas reglas están basadas en algoritmos matemáticos procedentes del ámbito de la economía empresarial y han sido definidos por expertos de la materia.

Estos cálculos han sido divididos en módulos según su temática para facilitar su tratamiento así como también para modularizar y encapsular a niveles de programación como ya se hizo en el proyecto SIMBA. Los módulos que se han creado son los siguientes:

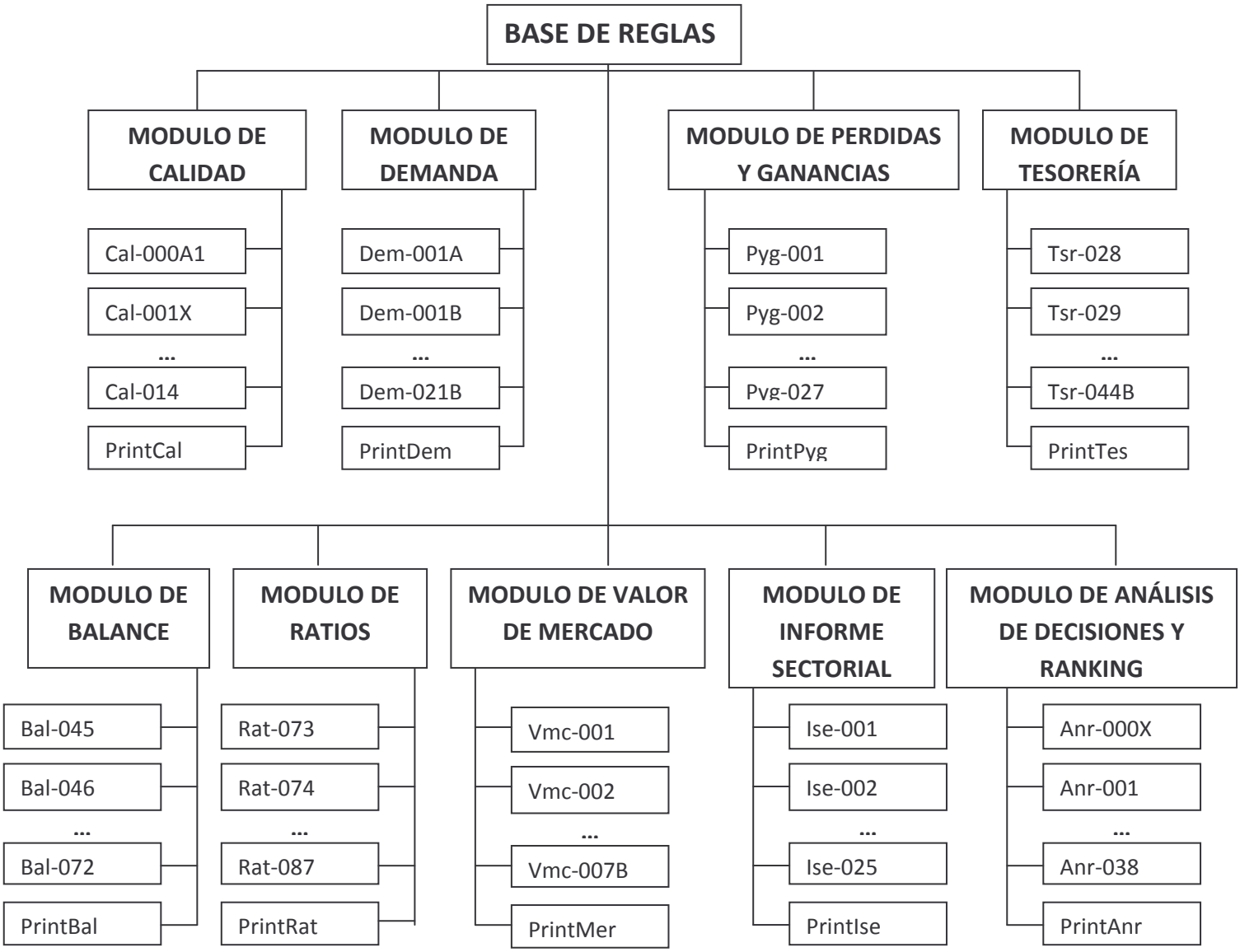


Figura 35: Representación de los diferentes grupos de reglas que forman el motor de razonamiento basado en reglas

Ahora vamos a presentar los tipos de regla que se han creado en el motor de razonamiento comenzando dentro del módulo de calidad, ya que nos servirá para extrapolarlas al resto de módulos del motor de razonamiento.

- **MODULO DE CALIDAD**: En este módulo se realizan todos los cálculos referentes a la calidad de las empresas. Se puede encontrar un desglose pormenorizado de los algoritmos realizados en cada cálculo en el ***ANEXO I***.

  Como ya se vio en el apartado *2.6* donde se explicaba la nomenclatura de la herramienta CLIPS, todas las reglas están compuestas por una parte derecha y una parte izquierda. Para realizar los cálculos de este módulo tenemos que asegurarnos de utilizar las variables adecuadas en la parte izquierda de la regla, seleccionando aquellas variables del historial que vamos a utilizar en los cálculos en la parte derecha de la regla. Podemos ilustrarlo con el siguiente ejemplo de cálculo relativamente sencillo:

  Tenemos el algoritmo del cálculo de calidad Cal-005 con resultado varMercadoI+D(t):

$$\text{var } MercadoI + D(t) = \left(\frac{I + DacumMercado(t)}{I + DacumMercado(t-1)} - 1 \right) x100$$

  Si lo implementamos en CLIPS sería de la siguiente manera:

```
(defrule Cal-005
        (declare (salience 476))
        (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
        ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po)
        (CAL002W ?calcW) (CAL002X ?calcX) (CAL002Y ?calcY) (CAL002Z ?calcZ))
        (contadorcal (cequipos ?ideo))
        =>
        (modify-instance ?res (CAL005W (* (/ (- ?calcW ?calcX) ?calcX) 100)))
        (modify-instance ?res (CAL005X (* (/ (- ?calcX ?calcY) ?calcY) 100)))
        (modify-instance ?res (CAL005Y (* (/ (- ?calcY ?calcZ) ?calcZ) 100)))
)
```

  Como podemos ver en este ejemplo, la parte izquierda de la regla viene separada de la parte derecha mediante la flecha "*=>*". También podemos observar como en la parte izquierda seleccionamos de la clase *RESULTADOSCALIDAD* las variables *CAL002W*, *CAL002X* y *CAL002Z* que luego utilizamos en la parte derecha para calcular *CAL005W*, *CAL005X* y *CAL005Y*.

Sin embargo no todos los cálculos son tan sencillos en este módulo de cálculo. Por ello, se han tenido que realizar algunas reglas preparatorias, cuyo fin es calcular variables auxiliares con sumatorio, que sirven para simplificar los cálculos de algunas reglas más complejas. Estas reglas auxiliares se realizan al principio del módulo y tienen el identificador *000XY* donde X es el segmento e Y es el periodo. Vamos a ver un ejemplo calculando entre otros el ∑ DID de empresas de segmento Alto en el periodo t-1:

```
(defrule Cal-000A1
      (declare (salience 496))
       (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
       (object (is-a HISTORIALAREACOMERCIAL) (IDEMPRESA ?ideo) (PERIODO ?pd) (RSEGMENTO "A"))
       (object (is-a HISTORIALDECISIONES) (IDEMPRESA ?ideo) (PERIODO ?po)
       (DID ?did) (DPRECIOEMP ?dprecioemp) (DPUBLICIDA ?dpublicida) (DPROMRED ?dpromed) (DFORMACION ?dformacion))
       ?res1 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 1) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       ?res2 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 2) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       ?res3 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 3) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       ?res4 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 4) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       ?res5 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 5) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       ?res6 <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA 6) (PERIODO ?po) (ESTADOCAL 1) (CAL001A1 ?cal001A)
       (CAL006A1 ?cal006A) (CAL006A1CONT ?cal006A1cont) (CAL008A1 ?cal008A) (CAL009A1 ?cal009A) (CAL011A1 ?cal011A))
       (contadorcal (cequipos ?ideo))
       (test (= (- ?po ?pd) 1))
      =>
       (modify-instance ?res1 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
       (modify-instance ?res2 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
       (modify-instance ?res3 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
       (modify-instance ?res4 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
       (modify-instance ?res5 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
       (modify-instance ?res6 (ESTADOCAL 2) (CAL001A1 (+ ?cal001A ?did)) (CAL006A1 (+ ?cal006A ?dprecioemp))
       (CAL008A1 (+ ?cal008A ?dpublicida)) (CAL009A1 (+ ?cal009A ?dpromed)) (CAL011A1 (+ ?cal011A ?dformacion))
       (CAL006A1CONT (+ ?cal006A1cont 1)))
)
```

Para explicar mejor este ejemplo vamos a analizar los datos de la Base de Conocimiento que utilizamos en la parte izquierda de esta regla:

Primero tenemos que saber de que periodo estamos realizando el sumatorio (*?po*). Esto lo conseguimos equiparando la instancia de la clase *EMPRESA* que contiene el periodo objetivo.

Después utilizamos los historiales de *HISTORIALAREACOMERCIAL* e *HISTORIALDECISIONES* porque son los datos que vamos a utilizar en el sumatorio. Pero no todos los periodos de estos historiales nos valen, y es por eso que utilizamos solo los de los periodos determinados *(?pd*) que posteriormente compararemos con el periodo objetivo (*?po*) mediante un "***test***" para que la diferencia entre ambos periodos sea la deseada. Tampoco nos sirven los datos de todos los equipos y es por eso que seleccionamos los datos que nos interesan filtrando las empresas de segmento Alto mediante *(**RSEGMENTO "A"**)* del *HISTORIALAREACOMERCIAL*. Esta es la manera en la que se filtra la información que nos interesa en el sumatorio.

Finalmente utilizamos los datos de los resultados de calidad *RESULTADOSCALIDAD* de todas las empresas (hay 6 entradas de *RESUTLADOSCALIDAD*, una por cada equipo).

Algo que también es particular a este módulo de calidad es que según los resultados que se vayan obteniendo entran en juego unos modificadores multiplicativos que se van generando a modo de decisiones. Algunos ejemplos de esto son estas reglas:

```
(defrule Deci-005
   (declare (salience 458))
   (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
   ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po)
   (CAL003Y ?calc3) (CAL005Y ?calc5))
   ?dec <- (object (is-a DECISIONES))
    (contadorcal (cequipos ?ideo))
    (test (>= ?calc3 ?calc5))
   =>
    (modify-instance ?dec (DECI005 0.1))
)
```

```
(defrule Deci-006
   (declare (salience 458))
   (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
  ?res <- (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po)
  (CAL004Z ?calc4) (CAL005Z ?calc5))
  ?dec <- (object (is-a DECISIONES))
   (contadorcal (cequipos ?ideo))
   (test (>= ?calc4 ?calc5))
  =>
   (modify-instance ?dec (DECI006 0.5))
)
```

En este caso estamos estableciendo decisiones dentro del módulo de calidad. Las decisiones pueden tener solamente 2 valores (uno negativo y otro positivo), que son definidos previamente en la Base del conocimiento. Para ahorrar proceso de razonamiento, al crear la Base de conocimiento y sus instancias, inicializamos las decisiones con su valor negativo. Después del proceso de razonamiento, solamente han cambiado su valor aquellas que cumplieron la parte izquierda de su regla correspondiente.

Para explicar mejor este ejemplo vamos a volver a analizar los datos de la Base de Conocimiento que utilizamos en la parte izquierda de esta regla:

Primero tenemos que saber de que empresa y de que periodo estamos decidiendo. Para ello como se explico anteriormente utilizamos la clase *EMPRESA* y sus valores de *IDEMPRESA* (*?ide*) y *PERIODO* (*?po*).

Después utilizamos los resultados de calidad *RESULTADOSCALIDAD* de la empresa que estamos tratando debido a que es en función de estos resultados de calidad a lo que se toma la decisión de modificar el valor de la decisión o no.

Finalmente, todos los módulos tienen una regla destinada a guardar en el fichero de salida todos los datos calculados. A modo de ejemplo representativo para todos los módulos, vamos a ver un ejemplo de este tipo de reglas:

```
(defrule imprimircalidad
   (declare (salience 452))
    (object (is-a EMPRESA) (NOMBREEMPRESA ?neo) (IDEMPRESA ?ideo) (PERIODO ?po))
    (object (is-a RESULTADOSCALIDAD) (IDEMPRESA ?ideo) (PERIODO ?po) (ESTADOCAL 48)
    (CAL001W ?cal001W) (CAL001X ?cal001X) (CAL001Y ?cal001Y) (CAL001Z ?cal001Z)
    (CAL002W ?cal002W) (CAL002X ?cal002X) (CAL002Y ?cal002Y) (CAL002Z ?cal002Z)
    (CAL003W ?cal003W) (CAL003X ?cal003X) (CAL003Y ?cal003Y) (CAL004W ?cal004W) (CAL004X ?cal004X) (CAL004Y ?cal004Y)
    (CAL005W ?cal005W) (CAL005X ?cal005X) (CAL005Y ?cal005Y) (CAL006W ?cal006W) (CAL006X ?cal006X) (CAL006Y ?cal006Y)
    (CAL007W ?cal007W) (CAL007X ?cal007X) (CAL007Y ?cal007Y) (CAL008W ?cal008W) (CAL008X ?cal008X) (CAL008Y ?cal008Y)
    (CAL009W ?cal009W) (CAL009X ?cal009X) (CAL009Y ?cal009Y) (CAL010W ?cal010W) (CAL010X ?cal010X) (CAL010Y ?cal010Y)
    (CAL011W ?cal011W) (CAL011X ?cal011X) (CAL011Y ?cal011Y) (CAL011Z ?cal011Z)
    (CAL012W ?cal012W) (CAL012X ?cal012X) (CAL012Y ?cal012Y) (CAL013W ?cal013W) (CAL013X ?cal013X) (CAL013Y ?cal013Y)
    (CAL014 ?cal014) (CAL014B ?cal014B))
   ?dec <- (object (is-a DECISIONES) (DECI003 ?dec003) (DECI004 ?dec004) (DECI005 ?dec005) (DECI006 ?dec006)
    (DECI007 ?dec007) (DECI008 ?dec008) (DECI009 ?dec009) (DECI010 ?dec010) (DECI011 ?dec011) (DECI012 ?dec012)
    (DECI013 ?dec013) (DECI014 ?dec014) (DECI015 ?dec015) (DECI016 ?dec016) (DECI017 ?dec017) (DECI018 ?dec018)
    (DECI019 ?dec019) (DECI020 ?dec020))
   ?con <- (contadorcal (cequipos ?ideo&: (<= ?ideo 6) ))
   =>
    (printout fs "Resulatdos Calidad para " ?neo crlf)
    (printout fs "---------------------------------" crlf)
    (printout fs "Calculo E" ?ideo "CAL001W: " ?cal001W crlf)
    (printout fs "Calculo E" ?ideo "CAL001X: " ?cal001X crlf)
    (printout fs "Calculo E" ?ideo "CAL001Y: " ?cal001Y crlf)
    (printout fs "Calculo E" ?ideo "CAL001Z: " ?cal001Z crlf)
    (printout fs crlf)
    (printout fs "Calculo E" ?ideo "CAL002W: " ?cal002W crlf)
    (printout fs "Calculo E" ?ideo "CAL002X: " ?cal002X crlf)
    (printout fs "Calculo E" ?ideo "CAL002Y: " ?cal002Y crlf)
    (printout fs "Calculo E" ?ideo "CAL002Z: " ?cal002Z crlf)
    (printout fs crlf)
            […]
    (printout fs "Calculo E" ?ideo "CAL014:  " ?cal014 crlf)
    (printout fs "Calculo E" ?ideo "CAL014B: " ?cal014B crlf)
    (printout fs crlf)
    (printout fs "Decisiones Calidad" crlf)
    (printout fs "Decision E" ?ideo "DCAL003:  " ?dec003 crlf)
            […]
    (printout fs "Decision E" ?ideo "DCAL020:  " ?dec020 crlf)
    (modify ?con (cequipos (+ ?ideo 1)))
)
```

Ahora vamos a describir brevemente el comportamiento del resto de módulos existentes:

- **MODULO DE DEMANDA**: En este módulo se realizan todos los cálculos referentes a la demanda de las empresas. Se puede encontrar un desglose pormenorizado de los algoritmos realizados en cada cálculo y en cada módulo que componen en el sistema dentro del ***ANEXO I.***

  Al igual que en los otras reglas, para realizar los cálculos de este módulo tenemos que asegurarnos de utilizar las variables adecuadas en la parte izquierda de la regla, seleccionando aquellas variables del historial que vamos a utilizar en los cálculos en la parte derecha de la regla.

- **MODULO DE PERDIDAS Y GANANCIAS**: En este módulo se realizan todos los cálculos referentes a las pérdidas y las ganancias de las empresas.

- **MODULO DE TESORERIA**: En este módulo se realizan todos los cálculos referentes a la tesorería de las empresas.

- **MODULO DE BALANCE**: En este módulo se realizan todos los cálculos referentes al balance financiero de las empresas.

- **MODULO DE RATIOS**: En este módulo se realizan todos los cálculos referentes a los ratios de las empresas.

- **MODULO DE VALOR DE MERCADO**: En este módulo se realizan todos los cálculos referentes al valor de mercado financiero de las empresas.

- **MODULO DE INFORME SECTORIAL**: En este módulo se realizan todos los cálculos referentes al informe sectorial de las empresas.

- **MODULO DE ANALISIS DE DECISIONES Y RANKING**: En este módulo se realizan todos los cálculos referentes al análisis financiero de decisiones de las empresas así como su ranking.

La implementación de todos los módulos ha sido muy compleja debido a la dificultad de codificar en CLIPS los algoritmos matemáticos del simulador SIMBA. Entre todos los módulos que componen el sistema, se realizan más de 500 cálculos en cada arbitraje, que se han codificado en más de 600 reglas mediante CLIPS.

Para elaborar todo el proyecto se han desarrollado más de 25.000 líneas de código en diferentes leguajes de programación (c, c++, SQL) y CLIPS.

4.4. Integración del motor de razonamiento CLIPS en SIMBA

Para integrar el motor de razonamiento basado en reglas en el simulador SIMBA ha sido necesario realizar conexiones y desconexiones (del antiguo motor) en varios archivos internos del sistema, así como conexiones con la página Web y la base de datos de SQL. Para que se vea más fácilmente las conexiones, se ha realizado un ejemplo de ejecución en el **Anexo I**.

Uno de los archivos más importantes e ilustrativos que se ha tenido que modificar para la integración del motor de razonamiento ha sido el fichero *GestionBDMercados.cpp*, que es llamado durante el arbitraje desde la página Web del simulador empresarial. Una modificación adicional que se ha realizado es la incorporación de comandos para la recogida de tiempos de cada motor mediante el comando *gettimeofday()*. En el archivo original de *GestionBDMercados.cpp* se llamaba a los diferentes módulos (modulo calidad, modulo demanda, etc.) por separado de la siguiente manera:

```
[...]
this->calcularEntornoProximoPeriodo(codSimulacion, nombreMercado, periodoActual);

gettimeofday(&ti, NULL);   // Instante inicial
//Motor de simulacion
ModuloCalidad *mc = new ModuloCalidad(conexionMysql, codSimulacion, nombreMercado, periodoActual);
mc->arbitrarModuloCalidad();
ModuloDemanda *md = new ModuloDemanda(conexionMysql, codSimulacion, nombreMercado, periodoActual);
md->arbitrarModuloDemanda();
                    [...]
ModuloValorMercadoContable *mv = new ModuloValorMercadoContable(conexionMysql, codSimulacion,
nombreMercado, periodoActual);
mv->arbitrarModuloValorMercadoContable();
mr->arbitrarModuloRatios2();
ModuloInformeSectorial *mie = new ModuloInformeSectorial(conexionMysql, codSimulacion, nombreMercado,
periodoActual);
mie->arbitrarModuloInformeSectorial();
ModuloAnalisisDecisiones *mad = new ModuloAnalisisDecisiones(conexionMysql, codSimulacion, nombreMercado,
periodoActual);
mad->arbitrarModuloAnalisisDecisiones();
ModuloRanking *mrk = new ModuloRanking(conexionMysql, codSimulacion, nombreMercado, periodoActual);
mrk->arbitrarModuloRanking();
gettimeofday(&tf, NULL);   // Instante final

tiempo=(tf.tv_sec - ti.tv_sec)+((tf.tv_usec - ti.tv_usec)/1000.0)*0.001;
printf("Has tardado: %g segundos\n", tiempo);

ftiempos = fopen("./MotorSimulacion/motorClips/tiempos/tiempoS.txt", "a");
if (ftiempos==NULL){  printf("No se pudo abrir el fichero de tiempos\n");}
sprintf(aux, "%g", tiempo);
fprintf(ftiempos, "%s\n", aux);
fclose(ftiempos);

//Eliminamos los objetos creados
                    [...]
```

Instante inicial

Antiguo Motor de razonamiento

Instante Final

Calculo del tiempo (Tiempo = final – inicial)

Metemos los tiempos en el fichero de tiempos

Como podemos observar, el motor que ya había implementado realizaba el arbitraje llamando a cada módulo de cálculo por separado y en diferentes ámbitos locales, a diferencia del nuevo motor de razonamiento basado en reglas que llama a un solo módulo general de cálculo (*gp->arbitrar();*) que realiza todas las operaciones en conjunto.

Por tanto, para la integración del motor de razonamiento basado en reglas se ha sustituido todas las llamadas que se realizaban anteriormente a cada módulo por un solo comando que llama al nuevo motor completo de la siguiente manera:

```
                                    [...]
this->calcularParametrosProximoPeriodo(codSimulacion, nombreMercado, periodoActual);
this->calcularEntornoProximoPeriodo(codSimulacion, nombreMercado, periodoActual);

//Motor de simulacion
GestorPredicciones *gp = new GestorPredicciones(conexionMysql, codSimulacion,
nombreMercado, periodoActual);

gettimeofday(&ti, NULL);   // Instante inicial

gp->arbitrar();

gettimeofday(&tf, NULL);   // Instante final
tiempo=(tf.tv_sec - ti.tv_sec)+((tf.tv_usec - ti.tv_usec)/1000.0)*0.001;
printf("Has tardado: %g segundos\n", tiempo);

ftiempos = fopen("./MotorSimulacion/motorClips/tiempos/tiempoS.txt", "a");
if (ftiempos==NULL){
  printf("No se pudo abrir el fichero de tiempos\n");
  eturn -1;
}

sprintf(aux, "%g", tiempo);
fprintf(ftiempos, "%s\n", aux);

fclose(ftiempos);
//Eliminamos los objetos creados
                                    [...]
```

Instante inicial

Nuevo Motor de razonamiento *

Instante Final

Calculo del tiempo (Tiempo = final – inicial)

Metemos los tiempos en el fichero de tiempos

* NOTA: Para realizar la llamada desde código c++ al motor de razonamiento en CLIPS, hemos creado un hilo (*p_thread*) que sustituye su ejecución por la de CLIPS mediante el comando *execvp()*. El padre espera a que el hijo (motor de CLIPS) termine su ejecución mediante *wait()*, para después meter los resultados en la BBDD del servidor mediante MYsql.

Capítulo 5

Resultados

Una vez que hemos explicado el diseño del motor de razonamiento, su ontología genérica y su base de reglas es hora de realizar pruebas de resultados, tiempos y rendimientos. En este capítulo se hace una recopilación de los resultados finales obtenidos (tiempos de ejecución) en una parte representativa de las pruebas realizadas. Se muestran los resultados generados y se proporciona un análisis de tiempos y rendimientos de cada uno de ellos.

5.1. Resultados Generados

En este apartado se da una visión sobre el formato de salida que tienen los resultados del motor de razonamiento basado en reglas. Un ejemplo de este formato es el módulo de calidad. Finalmente recordar que aunque se muestre en este formato, después de cada arbitraje la información se almacena en la base de datos del servidor en MySQL. De manera que se pueden consultar en cualquier momento los resultados obtenidos en el arbitraje de un periodo.

Predicciones para el periodo t= 1

Resultados Calidad para Equipo1

Calculo E1CAL001W: 1215009.820252
Calculo E1CAL001X: 1215006.070252
Calculo E1CAL001Y: 911254.552689
Calculo E1CAL001Z: 607503.035126

Calculo E1CAL002W: 4272963.157428
Calculo E1CAL002X: 4272955.657428
Calculo E1CAL002Y: 3204716.743071
Calculo E1CAL002Z: 2136477.828714

Calculo E1CAL003W: 0.00101752690799901
Calculo E1CAL003X: 0.0
Calculo E1CAL003Y: 0.0

Calculo E1CAL004W: 0.000308640433312586
Calculo E1CAL004X: 33.3333333333333
Calculo E1CAL004Y: 50.0

Calculo E1CAL005W: 0.000175522532909093
Calculo E1CAL005X: 33.3333333333334
Calculo E1CAL005Y: 50.0

Calculo E1CAL006W: 13.923447
Calculo E1CAL006X: 13.923447
Calculo E1CAL006Y: 13.923447

[…]

Calculo E1CAL013W: 0.0044567678570226
Calculo E1CAL013X: 33.3333333333333
Calculo E1CAL013Y: 50.0

Calculo E1CAL014: 5.609

Figura 52: Formato del fichero de salida de CLIPS con los resultados obtenidos en el arbitraje.

5.2. Simulaciones

En este apartado se muestra la información de los tiempos de ejecución recogidos en las simulaciones realizadas.

Las simulaciones se han realizado en un equipo AMD AM2 ATHLON de 64bits con 2Gb de memoria RAM.

Para realizar las pruebas de una manera más automatizada y a mayor escala se ha utilizado una aplicación llamada *Experimenter* (componente descrito en el apartado **2.4.Arquitectura Software de SIMBA** dentro de **Control de Simulación**) desarrollada por el laboratorio de investigación. Esta herramienta nos permite lanzar simulaciones en serie automáticamente, sin necesidad de lanzarlas una a una desde la página Web. Esto quiere decir que podemos decirle a la herramienta que lance automáticamente 5 simulaciones de 20 periodos en cada simulación. Esto hace un total de 100 arbitrajes seguidos que son lanzados en serie por esta aplicación. *Experimenter* también crea automáticamente a través de agentes inteligentes la toma de decisiones en cada periodo, de manera que no necesita que se introduzcan a mano desde la página Web. En definitiva, esta herramienta ha sido muy útil para la realización de pruebas conjuntas con los dos motores (antiguo y nuevo), que de manera no automatizada podían alargarse durante más de 32 horas.

Las pruebas se han realizado con 6 equipos en un entorno económico y de mercado común y competitivo. Cada una de las pruebas se compone de 10 simulaciones con 52 arbitrajes en cada simulación. Esto hace un total de 520 arbitrajes en cada prueba, lo que da una buena medida en cuanto a tiempos para comparar el rendimiento de cada motor.

Estas 10 pruebas que se van a exponer son solamente una pequeña muestra del gran tamaño de pruebas realizadas. Además precisamente se han seleccionado 10 pruebas en las que los tiempos hayan resultado estables (sin picos tanto a lo alto como a lo bajo). En otras pruebas, los resultados mostraban picos (en ocasiones bastante grandes) de tiempo tanto al alza como a la baja. Este suceso se puede explicar por el estado del servidor que recibe peticiones de varios motores instalados en el laboratorio del departamento de investigación. Las pruebas realizadas en horas nocturnas (entre 00:00 y 8:00), han dado como resultado tiempos bastante bajo de media, si los comparamos con las pruebas realizadas en horas diurnas.

Una vez expuestas todas las bases sobre las que se han realizado las pruebas, pasamos a exponer una muestra de simulaciones en los siguientes subapartados.

5.2.1. Prueba 1

Después de realizar esta primera simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	6,01109 seg.	SIMULACION 1	78,85288019 seg.
SIMULACION 2	6,35115 seg.	SIMULACION 2	76,40905769 seg.
SIMULACION 3	6,25362 seg.	SIMULACION 3	78,19359231 seg.
SIMULACION 4	7,18908 seg.	SIMULACION 4	75,06419231 seg.
SIMULACION 5	5,54925 seg.	SIMULACION 5	69,50610385 seg.
SIMULACION 6	5,54669 seg.	SIMULACION 6	81,54254423 seg.
SIMULACION 7	5,55070 seg.	SIMULACION 7	88,52395192 seg.
SIMULACION 8	5,58419 seg.	SIMULACION 8	81,67337885 seg.
SIMULACION 9	5,59120 seg.	SIMULACION 9	79,48343462 seg.
SIMULACION 10	5,87914 seg.	SIMULACION 10	78,98290596 seg.
MEDIA PRUEBA 1	**5,95061 seg.**	**MEDIA PRUEBA 1**	**78,82320419 seg.**

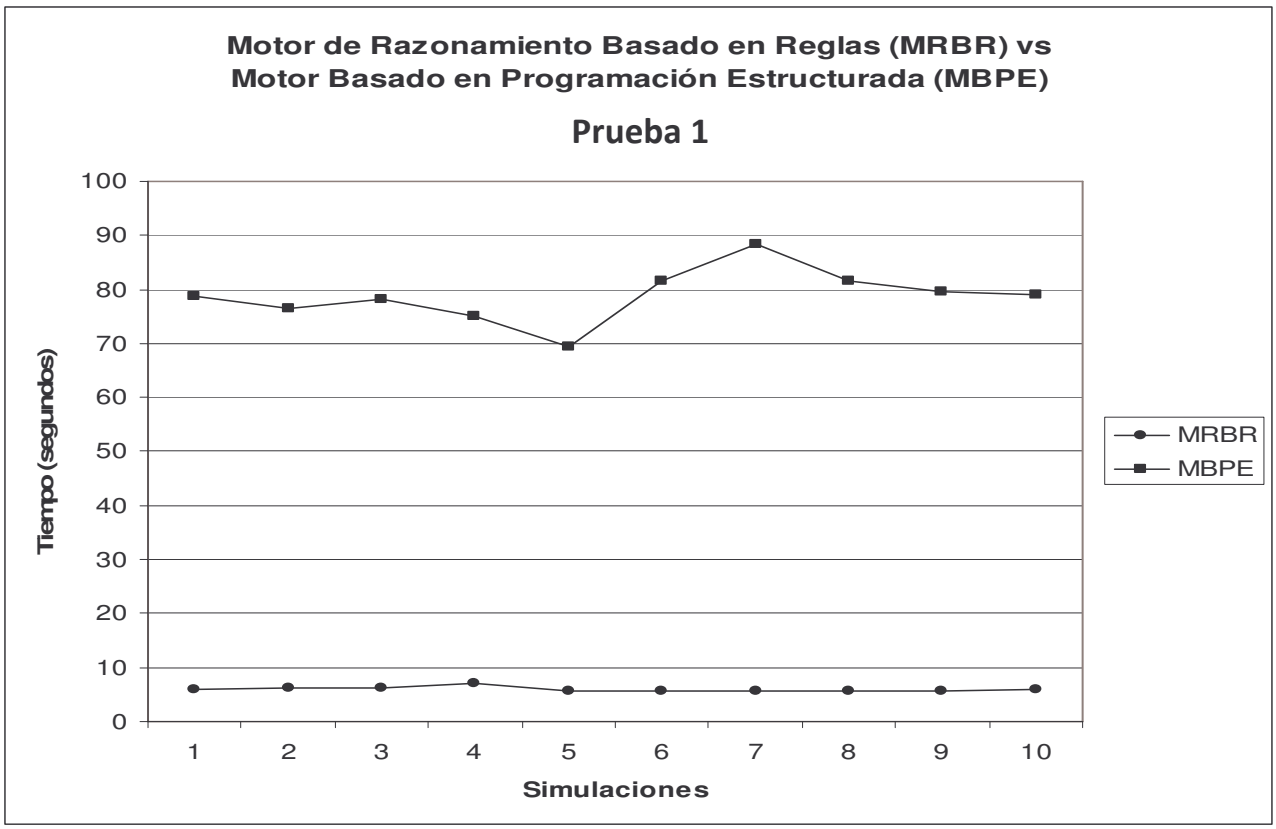


Figura 53: Gráfica de tiempos para la prueba 1.

5.2.2. Prueba 2

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,13814 seg.	SIMULACION 1	62,39947212 seg.
SIMULACION 2	5,10313 seg.	SIMULACION 2	72,06946154 seg.
SIMULACION 3	5,10434 seg.	SIMULACION 3	72,27517885 seg.
SIMULACION 4	5,08506 seg.	SIMULACION 4	67,65409750 seg.
SIMULACION 5	5,10201 seg.	SIMULACION 5	67,66256577 seg.
SIMULACION 6	5,19660 seg.	SIMULACION 6	67,67601250 seg.
SIMULACION 7	6,32565 seg.	SIMULACION 7	70,05551942 seg.
SIMULACION 8	6,63480 seg.	SIMULACION 8	71,06757250 seg.
SIMULACION 9	5,13898 seg.	SIMULACION 9	70,22046442 seg.
SIMULACION 10	4,96948 seg.	SIMULACION 10	77,82888942 seg.
MEDIA PRUEBA 2	**5,37982 seg.**	**MEDIA PRUEBA 2**	**69,89092340 seg.**

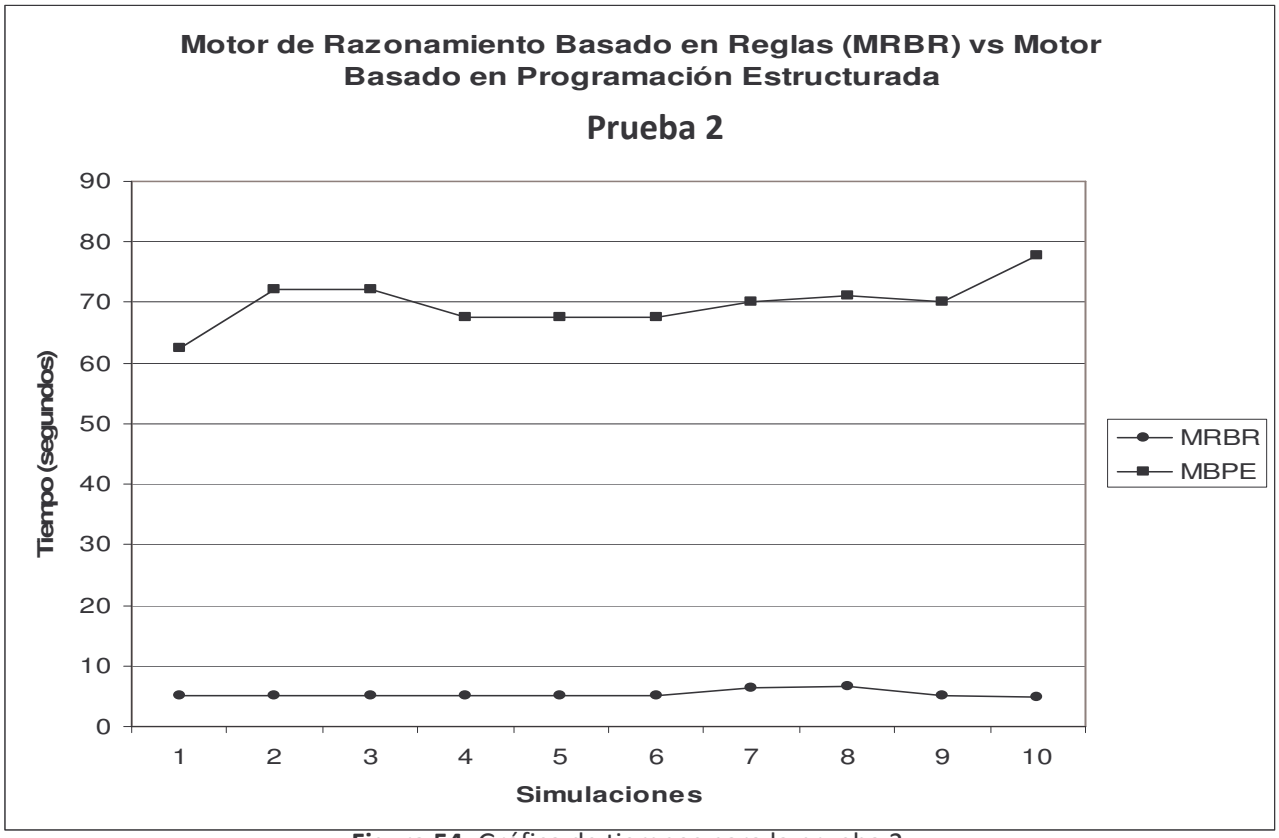


Figura 54: Gráfica de tiempos para la prueba 2.

5.2.3. Prueba 3

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,09189 seg.	SIMULACION 1	68,20050615 seg.
SIMULACION 2	4,03877 seg.	SIMULACION 2	68,09422865 seg.
SIMULACION 3	5,15526 seg.	SIMULACION 3	68,04253288 seg.
SIMULACION 4	5,17407 seg.	SIMULACION 4	68,05310788 seg.
SIMULACION 5	5,17831 seg.	SIMULACION 5	68,05058635 seg.
SIMULACION 6	5,18639 seg.	SIMULACION 6	68,05869596 seg.
SIMULACION 7	5,19949 seg.	SIMULACION 7	68,08582981 seg.
SIMULACION 8	5,30925 seg.	SIMULACION 8	69,25468750 seg.
SIMULACION 9	5,25150 seg.	SIMULACION 9	68,10778769 seg.
SIMULACION 10	5,70516 seg.	SIMULACION 10	61,30630115 seg.
MEDIA PRUEBA 3	**5,12901 seg.**	**MEDIA PRUEBA 3**	**67,5254264 seg.**

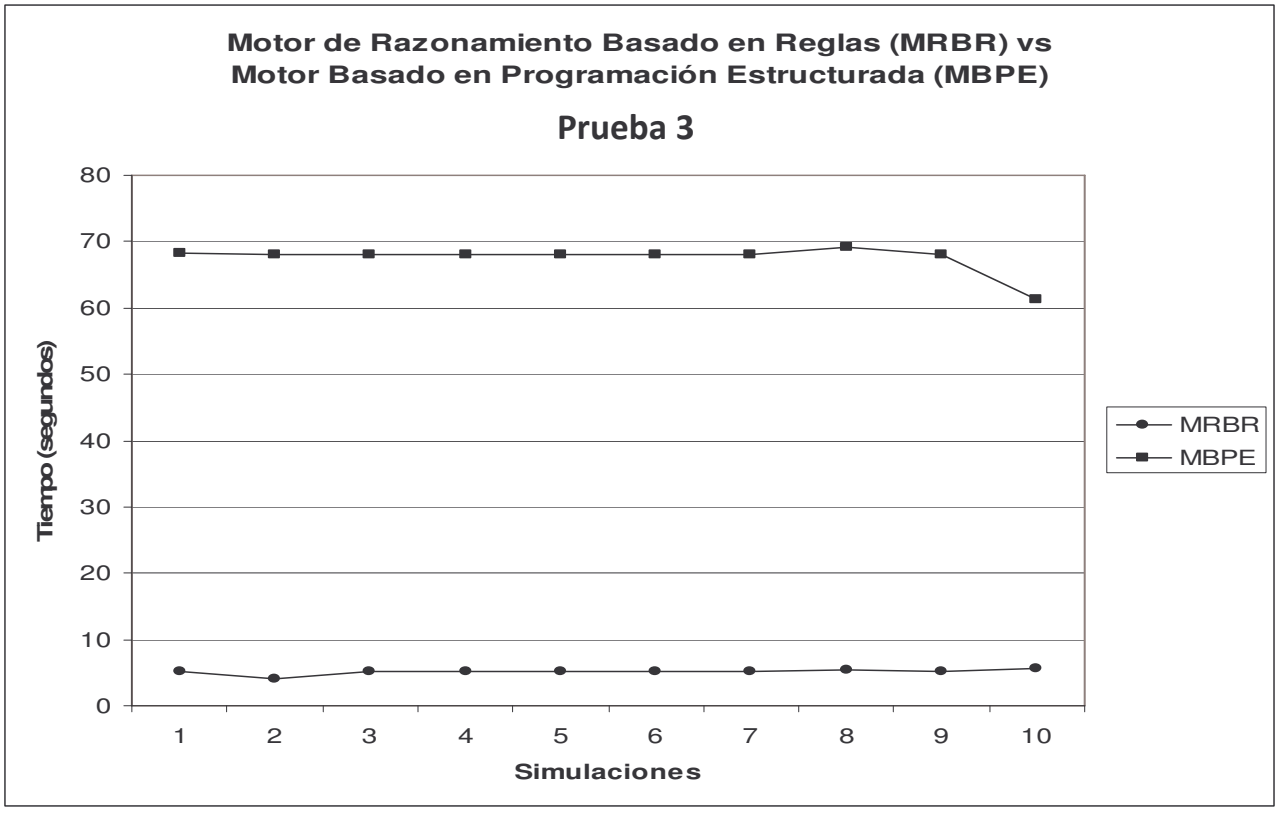


Figura 55: Gráfica de tiempos para la prueba 3.

5.2.4. Prueba 4

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,64575 seg.	SIMULACION 1	67,28502 seg.
SIMULACION 2	5,16238 seg.	SIMULACION 2	67,47809 seg.
SIMULACION 3	5,15373 seg.	SIMULACION 3	67,21205 seg.
SIMULACION 4	5,19553 seg.	SIMULACION 4	67,26910 seg.
SIMULACION 5	5,18856 seg.	SIMULACION 5	67,30661 seg.
SIMULACION 6	5,13294 seg.	SIMULACION 6	67,34540 seg.
SIMULACION 7	5,09597 seg.	SIMULACION 7	67,34557 seg.
SIMULACION 8	5,15476 seg.	SIMULACION 8	67,40468 seg.
SIMULACION 9	5,17768 seg.	SIMULACION 9	67,47664 seg.
SIMULACION 10	5,27265 seg.	SIMULACION 10	70,89949 seg.
MEDIA PRUEBA 4	**5,21800 seg.**	**MEDIA PRUEBA 4**	**67,70226 seg.**

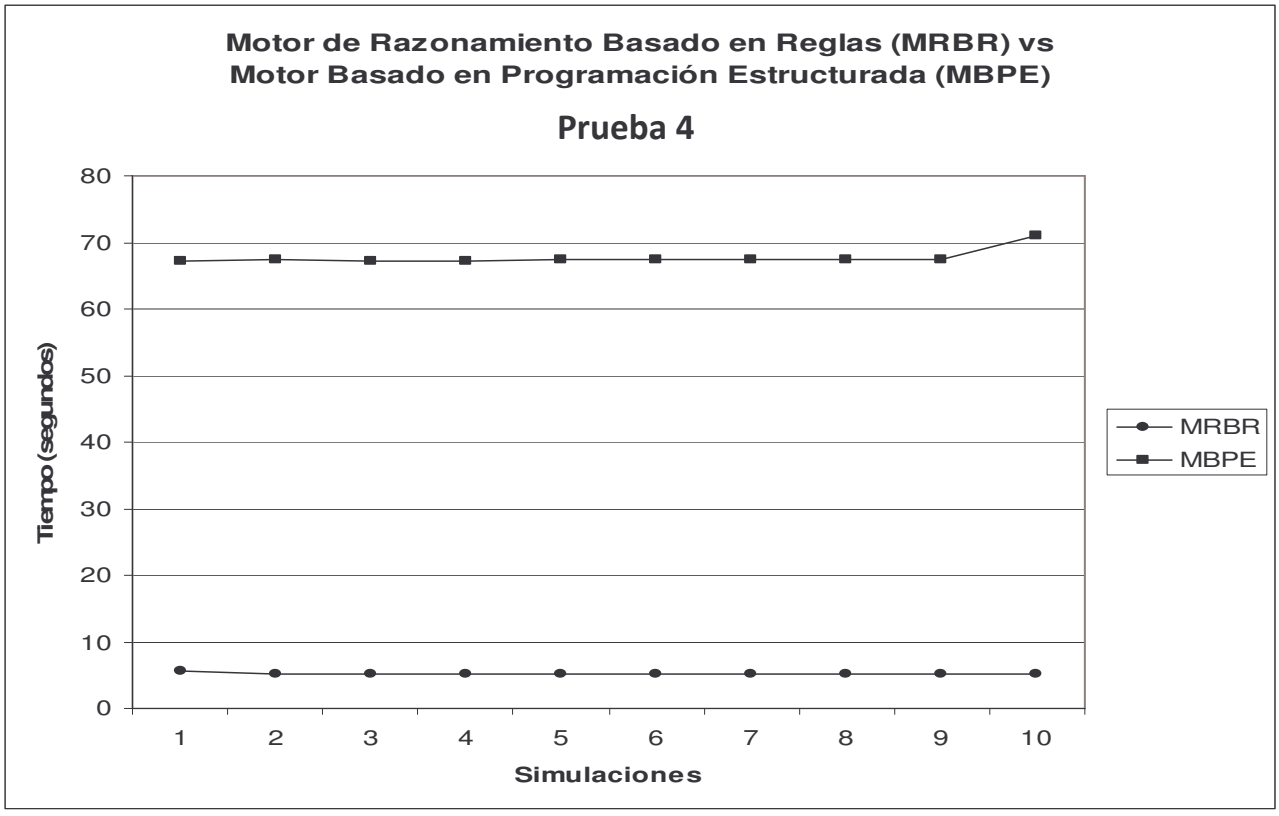


Figura 56: Gráfica de tiempos para la prueba 4.

5.2.5. Prueba 5

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,18269 seg.	SIMULACION 1	67,66246 seg.
SIMULACION 2	5,05000 seg.	SIMULACION 2	67,65071 seg.
SIMULACION 3	5,04717 seg.	SIMULACION 3	67,66314 seg.
SIMULACION 4	5,08097 seg.	SIMULACION 4	67,67409 seg.
SIMULACION 5	5,05872 seg.	SIMULACION 5	68,40063 seg.
SIMULACION 6	5,07120 seg.	SIMULACION 6	67,72479 seg.
SIMULACION 7	5,06740 seg.	SIMULACION 7	67,64273 seg.
SIMULACION 8	5,11959 seg.	SIMULACION 8	67,66418 seg.
SIMULACION 9	5,11077 seg.	SIMULACION 9	67,70760 seg.
SIMULACION 10	5,24220 seg.	SIMULACION 10	70,95805 seg.
MEDIA PRUEBA 5	**5,10307 seg.**	**MEDIA PRUEBA 5**	**68,07484 seg.**

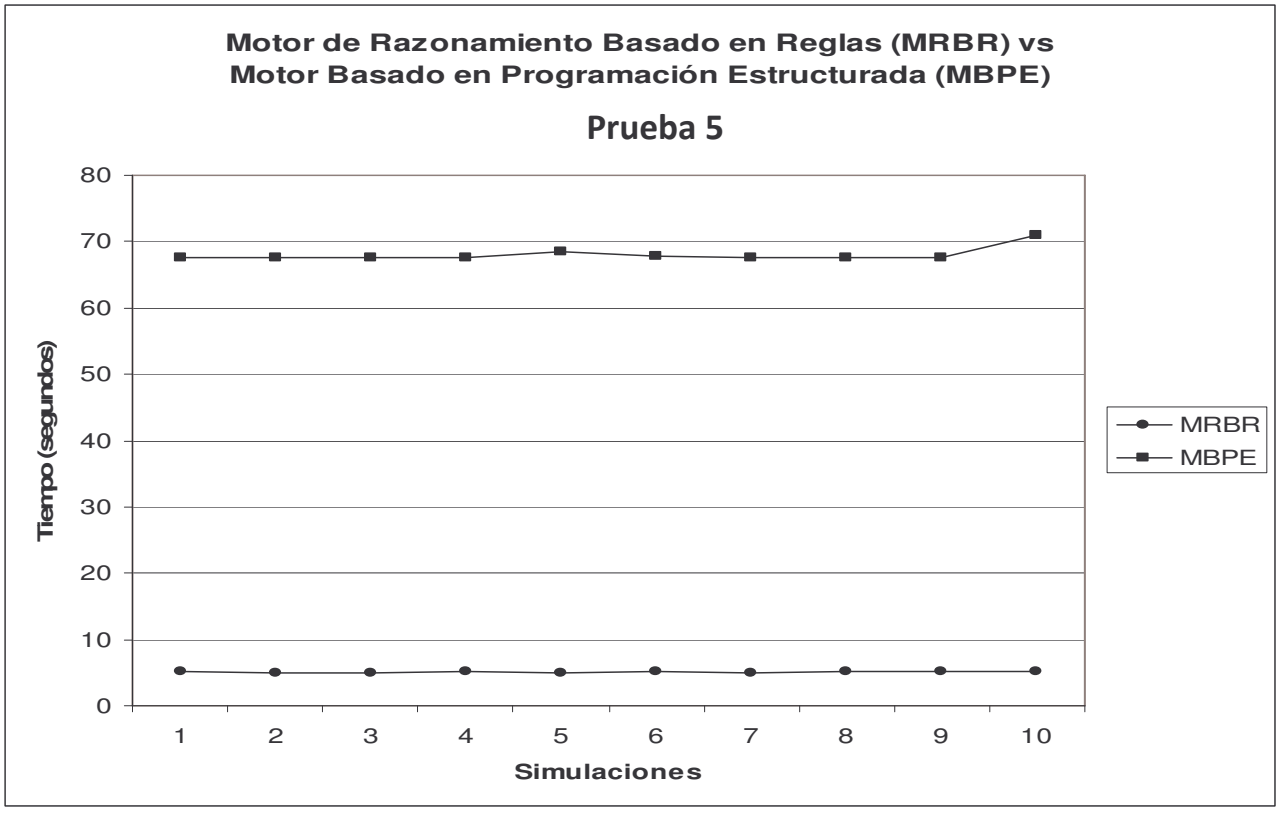


Figura 57: Gráfica de tiempos para la prueba 5.

5.2.6. Prueba 6

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,41226 seg.	SIMULACION 1	55,40038 seg.
SIMULACION 2	7,48824 seg.	SIMULACION 2	66,83618 seg.
SIMULACION 3	7,61852 seg.	SIMULACION 3	66,99589 seg.
SIMULACION 4	6,29691 seg.	SIMULACION 4	67,87903 seg.
SIMULACION 5	5,07634 seg.	SIMULACION 5	66,20679 seg.
SIMULACION 6	5,06815 seg.	SIMULACION 6	67,95749 seg.
SIMULACION 7	5,06991 seg.	SIMULACION 7	70,46225 seg.
SIMULACION 8	5,08181 seg.	SIMULACION 8	75,14700 seg.
SIMULACION 9	5,11907 seg.	SIMULACION 9	67,74183 seg.
SIMULACION 10	5,30158 seg.	SIMULACION 10	75,85961 seg.
MEDIA PRUEBA 6	**5,75328 seg.**	**MEDIA PRUEBA 6**	**68,04864 seg.**

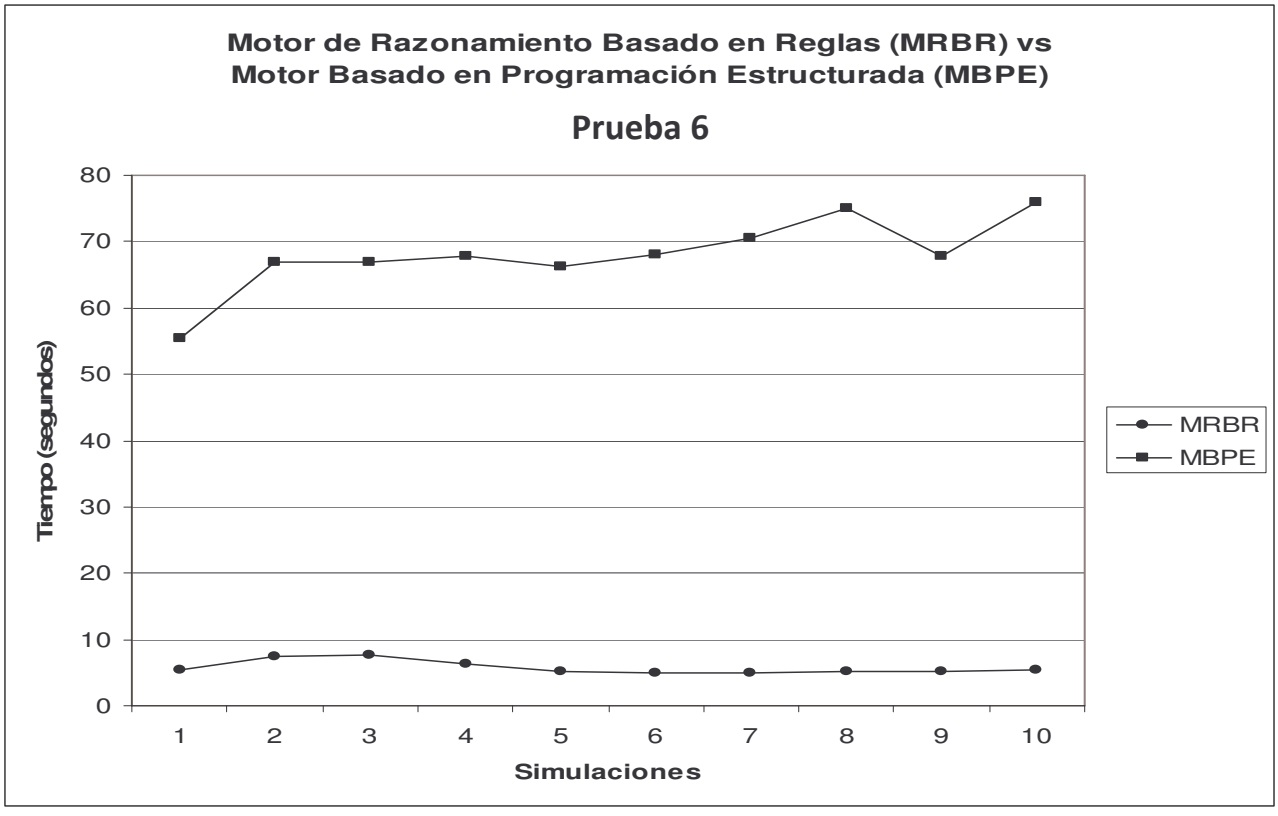


Figura 58: Gráfica de tiempos para la prueba 6.

5.2.7. Prueba 7

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,280326538 seg.	SIMULACION 1	69,28420 seg.
SIMULACION 2	5,138244038 seg.	SIMULACION 2	69,45441 seg.
SIMULACION 3	5,121231538 seg.	SIMULACION 3	69,44635 seg.
SIMULACION 4	5,147664231 seg.	SIMULACION 4	64,56516 seg.
SIMULACION 5	5,129201923 seg.	SIMULACION 5	70,41944 seg.
SIMULACION 6	5,167718269 seg.	SIMULACION 6	69,43980 seg.
SIMULACION 7	5,144965000 seg.	SIMULACION 7	69,53189 seg.
SIMULACION 8	5,103414615 seg.	SIMULACION 8	72,46330 seg.
SIMULACION 9	5,200559423 seg.	SIMULACION 9	65,21930 seg.
SIMULACION 10	5,246726538 seg.	SIMULACION 10	80,37988 seg.
MEDIA PRUEBA 7	**5,168005212 seg.**	**MEDIA PRUEBA 7**	**70,02037 seg.**

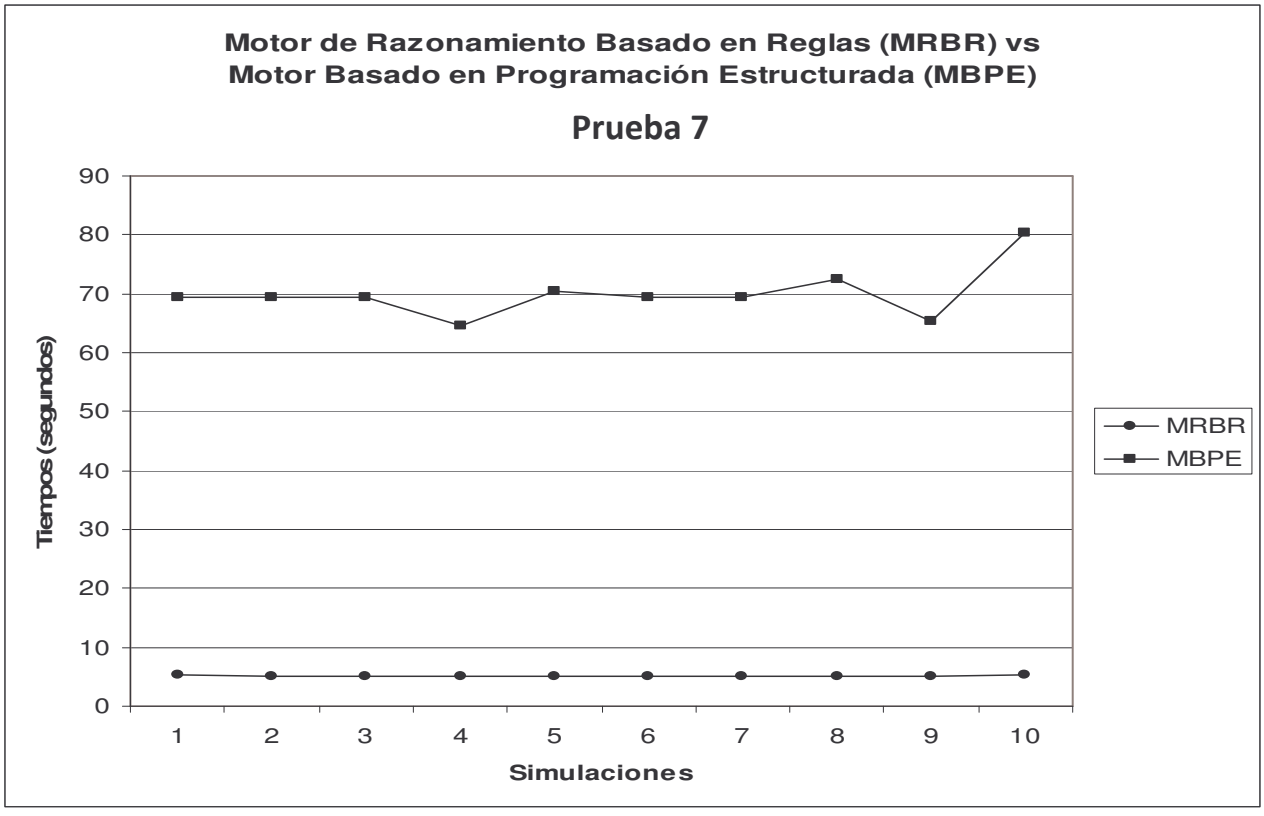


Figura 59: Gráfica de tiempos para la prueba 7.

5.2.8. Prueba 8

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,177290577 seg.	SIMULACION 1	68,40037769 seg.
SIMULACION 2	5,041403077 seg.	SIMULACION 2	66,83618096 seg.
SIMULACION 3	5,071422308 seg.	SIMULACION 3	66,99588519 seg.
SIMULACION 4	5,063061154 seg.	SIMULACION 4	67,87903288 seg.
SIMULACION 5	5,048993462 seg.	SIMULACION 5	66,20678635 seg.
SIMULACION 6	5,076892501 seg.	SIMULACION 6	67,95749096 seg.
SIMULACION 7	5,096568077 seg.	SIMULACION 7	70,46224692 seg.
SIMULACION 8	5,096502885 seg.	SIMULACION 8	70,14699827 seg.
SIMULACION 9	5,159423269 seg.	SIMULACION 9	67,74182538 seg.
SIMULACION 10	5,235902506 seg.	SIMULACION 10	70,85960501 seg.
MEDIA PRUEBA 8	**5,106745981 seg.**	**MEDIA PRUEBA 8**	**68,34864296 seg.**

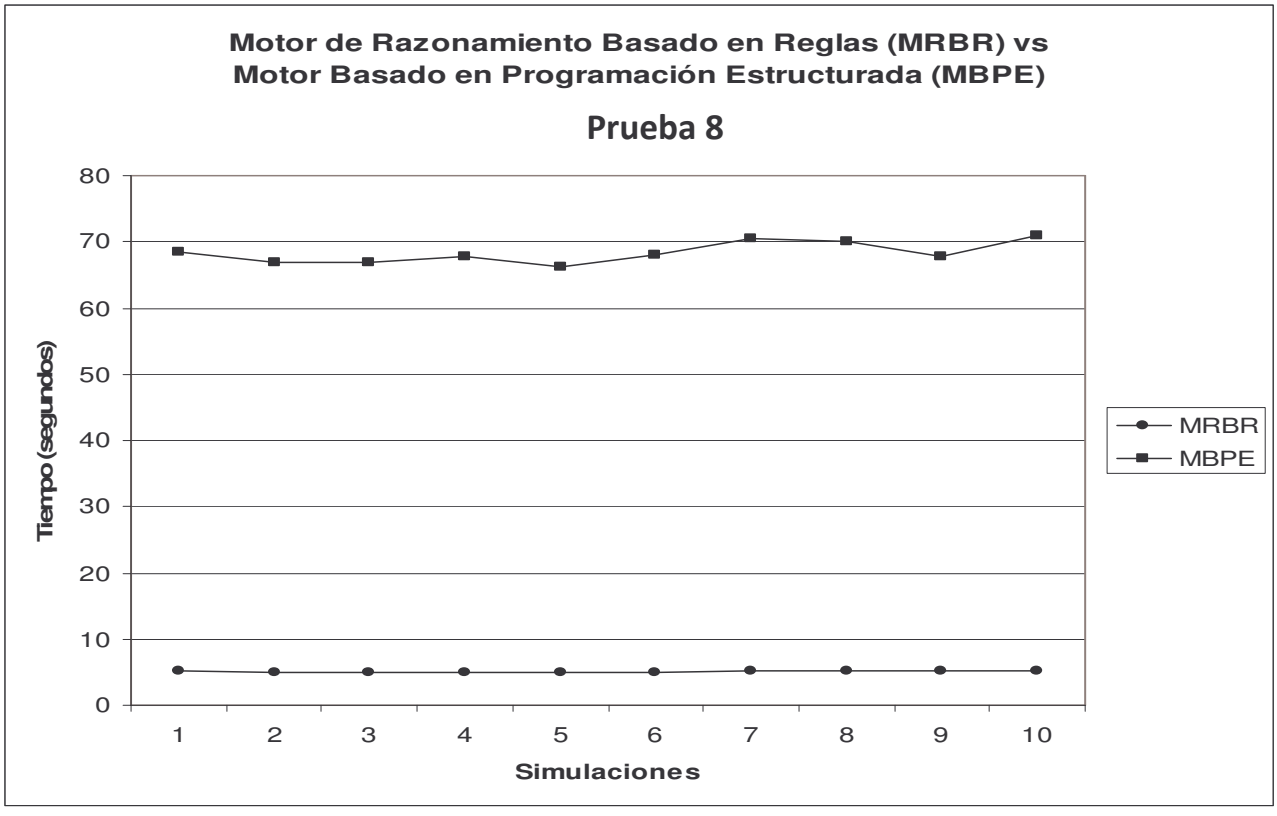


Figura 60: Gráfica de tiempos para la prueba 8.

5.2.9. Prueba 9

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,264879532 seg.	SIMULACION 1	68,75652155 seg.
SIMULACION 2	5,124679538 seg.	SIMULACION 2	66,78962146 seg.
SIMULACION 3	5,146794257 seg.	SIMULACION 3	64,99588519 seg.
SIMULACION 4	5,021976534 seg.	SIMULACION 4	69,66314848 seg.
SIMULACION 5	5,217946232 seg.	SIMULACION 5	70,58909143 seg.
SIMULACION 6	5,226884601 seg.	SIMULACION 6	71,54898842 seg.
SIMULACION 7	5,046568075 seg.	SIMULACION 7	70,48963643 seg.
SIMULACION 8	5,026502888 seg.	SIMULACION 8	69,65684012 seg.
SIMULACION 9	5,059423275 seg.	SIMULACION 9	71,54985876 seg.
SIMULACION 10	5,178916455 seg.	SIMULACION 10	70,48995015 seg.
MEDIA PRUEBA 9	**5,131457138 seg.**	**MEDIA PRUEBA 9**	**69,45295548 seg.**

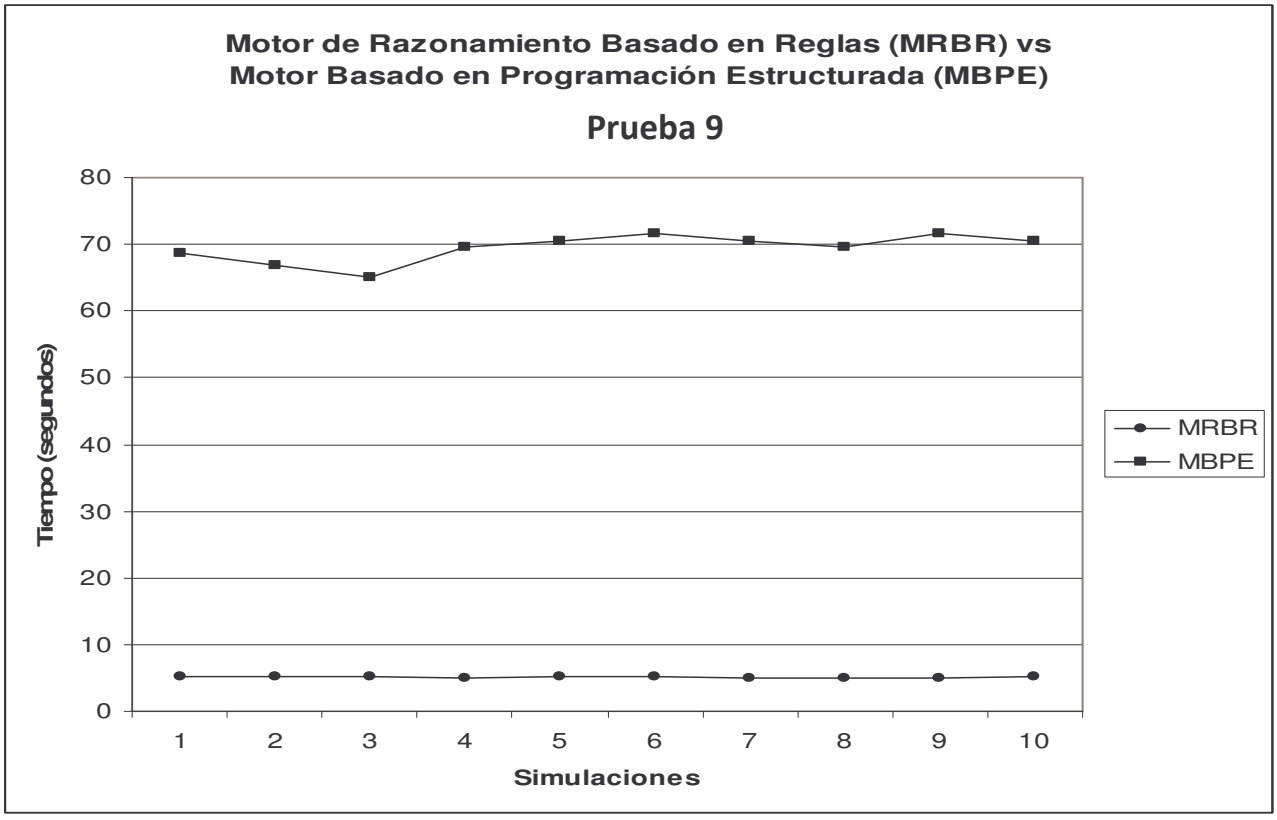


Figura 61: Gráfica de tiempos para la prueba 9.

5.2.10. Prueba 10

Después de realizar esta simulación se han registrado la siguiente media de tiempos para cada motor del simulador empresarial SIMBA:

MOTOR DE RAZONAMIENTO BASADO EN REGLAS		MOTOR BASADO EN PROGRAMACIÓN ESTRUCTURADA	
SIMULACION 1	5,067958135 seg.	SIMULACION 1	71,49765312 seg.
SIMULACION 2	5,026798315 seg.	SIMULACION 2	70,79864328 seg.
SIMULACION 3	5,087953489 seg.	SIMULACION 3	68,13267983 seg.
SIMULACION 4	5,064379252 seg.	SIMULACION 4	69,54796315 seg.
SIMULACION 5	5,025679138 seg.	SIMULACION 5	67,18795346 seg.
SIMULACION 6	5,041379562 seg.	SIMULACION 6	69,87946235 seg.
SIMULACION 7	5,013264973 seg.	SIMULACION 7	71,79564313 seg.
SIMULACION 8	5,109462346 seg.	SIMULACION 8	70,46732619 seg.
SIMULACION 9	5,121346081 seg.	SIMULACION 9	69,48795613 seg.
SIMULACION 10	5,110480884 seg..	SIMULACION 10	70,59765846 seg.
MEDIA PRUEBA 10	**5,066870217seg.**	**MEDIA PRUEBA 10**	**69,93929391 seg.**

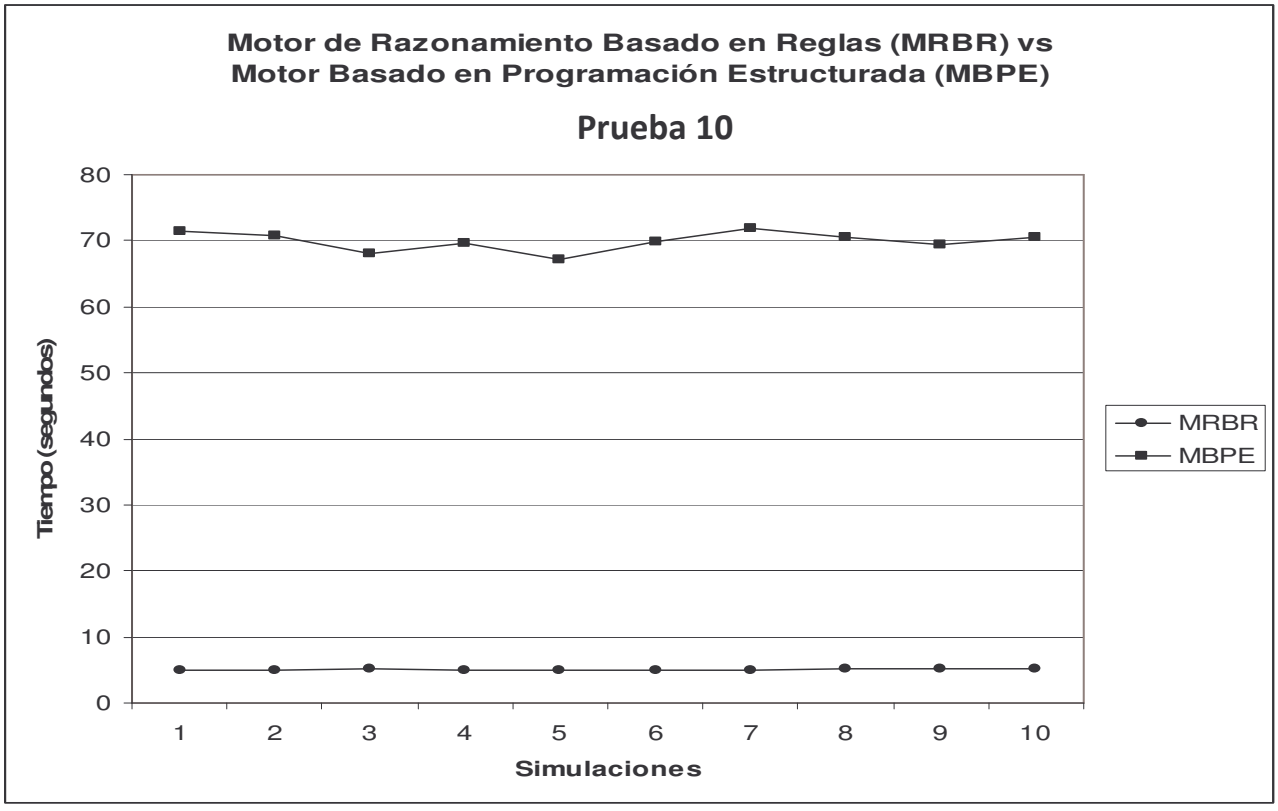


Figura 62: Gráfica de tiempos para la prueba 10.

5.3. Evaluación de los resultados del proyecto

En función de los resultados obtenidos, en este apartado vamos a realizar una evaluación del rendimiento de cada motor y de la mejora considerable que ha supuesto el motor de razonamiento basado en reglas, desarrollado en este proyecto para el simulador empresarial SIMBA.

Podríamos considerar el *Speedup* (S) como el tiempo del motor basado en programación estructurada con respecto al motor de razonamiento basado en reglas, de tal manera que comparándolos podamos observar el incremento en porcentaje de sus prestaciones.

S = tMBPE / tMRBR

La media de todos los tiempos de ejecución obtenidos en las simulaciones del motor basado en programación estructurada (tMBPE) es igual a 77,3442 segundos. Por otro lado, la media de todos los tiempos de ejecución obtenidos en las simulaciones del motor de razonamiento basado en reglas (tMRBR) es igual a 5,42230 segundos. De esta manera obtenemos que:

S = tMBPE / tMRBR = 77,34422 / 5,42230 = 14,26410658

Esto quiere decir que en el tiempo en el que antes el motor basado en programación estructurada realizaba un arbitraje, ahora el nuevo motor puede realizar 14,26 arbitrajes, lo que conlleva una gran mejora. Hablando en términos de eficiencia (E) podríamos calcular que:

E = tMRBR * 100 / tMBPE = 7,010603816

O lo que es lo mismo, es un **701,06%** más eficiente en cuanto tiempo. Finalmente en aspectos de eficiencia, podemos concluir que la realización del motor de razonamiento basado en reglas para el simulador empresarial SIMBA ha supuesto una gran mejora.

No podemos terminar esta reflexión de rendimiento y eficiencia sin recordar que como se mencionó en el apartado **4.2.Modelo de conocimiento**, hay bastantes factores que determinan en gran medida el tiempo de ejecución de los motores. En este proyecto, por ejemplo, ha sido determinante el número de accesos a la base de datos que se realiza. Este número y tipo de accesos son bastante más eficientes, en comparación a los que se realizaban el anterior motor basado en programación estructurada. En el anterior motor de razonamiento, cada vez que se necesitaba un dato, se realizaba un acceso a la base de datos. En cambio en el nuevo motor se genera un fichero local con los datos que vamos a trabajar, para evitar accesos innecesarios al servidor Web que aloja la base de datos, y los desperdicios de tiempo que ello conlleva. Este hecho puede explicar de manera razonable la gran diferencia de tiempos que hay entre los dos motores.

Capítulo 6

Gestión del Proyecto

En este apartado vamos a detallar los aspectos referentes a la gestión del proyecto tales como los modelos seguidos, tiempo y recursos utilizados, costes y decisiones importantes.

6.1. Metodología

Uno de los problemas que se tuvieron al inicio del proyecto tenía que ver con la gestión del mismo. Este problema surgió porque, debido a que estamos trabajando con un SBC, no estaba claro si había que utilizar una metodología orientada a la Ingeniería del Conocimiento, o por el contrario utilizar una metodología orientada a la Ingeniería del Software como un software común. Para aclarar como se tomó la decisión de utilizar una metodología u otra, es necesario explicar algunos conceptos antes.

La ingeniería del conocimiento es el proceso de adquirir, estructurar, formalizar y hacer operativo un conjunto de conocimientos para formar un Sistema Basado en el Conocimiento (SBC). Para construir un SBC hay que seguir las siguientes fases:

1. Identificación del problema
2. Adquisición del conocimiento
3. Conceptualización del SBC
4. Formalización SBC
5. Validación del SBC

Cuando se desarrolló el simulador empresarial SIMBA ya se disponía de todos los algoritmos económicos necesarios para desarrollar el sistema, por lo que no hizo falta desarrollar una metodología acorde a la Ingeniería del conocimiento.

Finalmente se decidió utilizar una metodología orientada a la Ingeniería del Software porque no era necesario hacer las fases de Adquisición del conocimiento, Conceptualización y Formalización, ya que se podían reutilizar de la documentación del proyecto SIMBA.

6.2. Ciclo de Vida

El proyecto ha sido diseñado siguiendo un ciclo de vida en cascada formado por las siguientes fases:

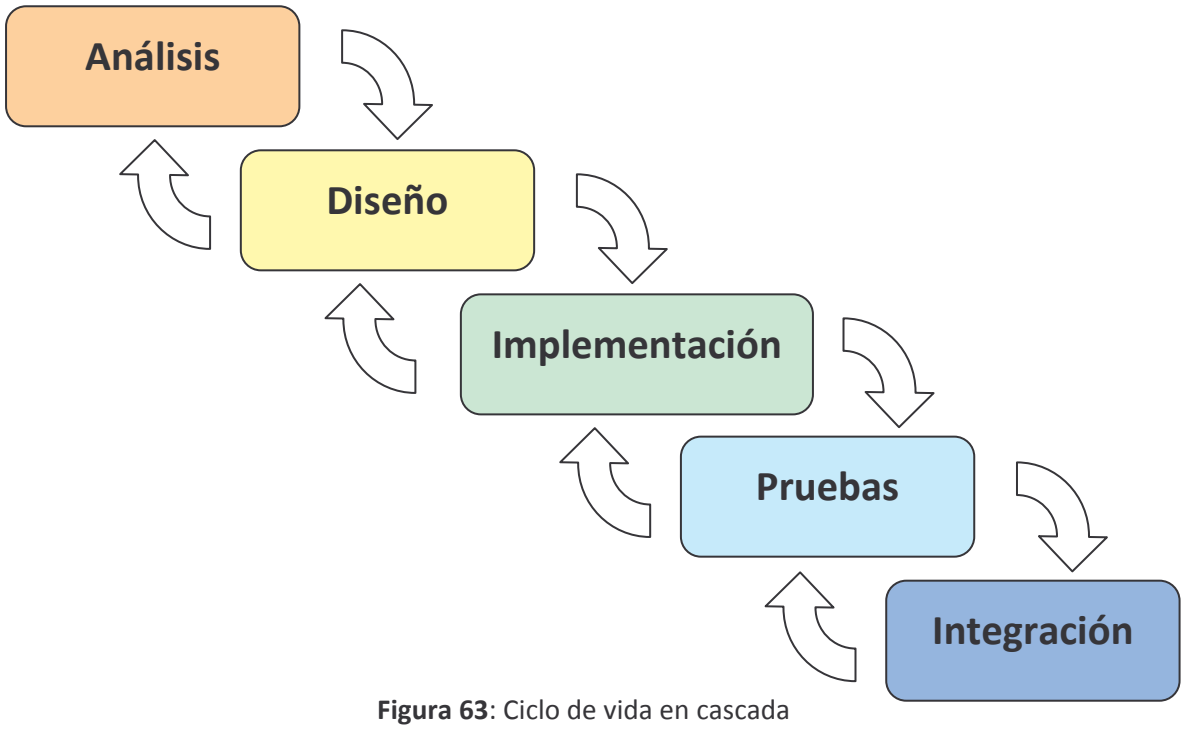


Figura 63: Ciclo de vida en cascada

Este modelo admite la posibilidad de hacer iteraciones. Esto quiere decir que si se tiene que volver a una anterior porque se hizo mal algún diseño o alguna implementación, se puede hacer, pero para avanzar el proyecto hay que recorrer de nuevo el resto de las etapas para asegurar una buena gestión del proyecto.

Después de cada etapa se realiza una revisión para comprobar si se puede pasar a la siguiente etapa.

Las ventajas del ciclo de vida en cascada son:

- La planificación es sencilla.
- La calidad del producto resultante es alta.
- Permite trabajar con personal poco cualificado.

Los inconvenientes de este tipo de ciclo de vida son:

- La necesidad de tener todos los requisitos al principio. Lo normal es que el cliente no tenga perfectamente definidas las especificaciones del sistema, o puede ser que surjan necesidades imprevistas.
- No se tiene el producto hasta el final, esto quiere decir que el cliente puede impacientarse

Los tipos de proyectos para los que el ciclo de vida en cascada es adecuado son:

- Se está desarrollando un tipo de producto que no es novedoso.
- Aquellos para los que se dispone de todas las especificaciones desde el principio, por ejemplo los de reingeniería como es el caso, ya que el sistema SIMBA ya existía con anterioridad y surgió la necesidad de cambiar su motor de razonamiento.
- Proyectos complejos que se entienden bien desde el principio.

Se ha elegido un ciclo de vida software en cascada porque, como se ha dicho, es altamente adecuado para este tipo de proyecto, ya que el modelo se adapta perfectamente a la necesidad del desarrollo del motor de razonamiento (proyecto de reingeniería). Además el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior, de manera que si se detectan problemas en una etapa, rápidamente se puede volver a la etapa anterior y solucionar los problemas para que no repercutan al resto del proyecto software.

El tiempo dedicado a cada una de las fases del proyecto se recoge en la siguiente tabla:

Fases	Tiempo (Nº horas)
Análisis	70
Diseño	45
Implementación	170
Pruebas	145
Integración	75
Total Proyecto	505

Sin embargo, este desglose es muy general porque cada fase se ha dividido en tareas más pequeñas y específicas de la siguiente manera:

Las tareas de la fase de análisis, así como el tiempo invertido en cada una son los siguientes:

Tareas de Análisis	Tiempo (Nº horas)
Documentación SIMBA	10
Análisis	45
Generación Memoria Análisis	15
Total Análisis	70

Las tareas de la fase de diseño, así como el tiempo invertido en cada una son los siguientes:

Tareas de Diseño	Tiempo (Nº horas)
Diseño Ontología Genérica en CLIPS	8
Diseño Historial en CLIPS	10
Diseño Base de Reglas en CLIPS	10
Diseño conexión con Mysql	5
Diseño Generación automática de Historial	5
Generación Memoria Diseño	7
Total Diseño	45

Las tareas de la fase de implementación, así como el tiempo invertido en cada una son

Tareas de Implementación	Tiempo (Nº horas)
Recuperación de Historial del Servidor	15
Módulo de Calidad	60
Módulo de Demanda	20
Módulo de Cuenta de Pérdidas y Ganancias	10
Módulo de Tesorería	10
Módulo de Balance	15
Módulo de Ratios	7
Módulo de Valor de Mercado	5
Módulo de Informe Sectorial	5
Módulo de Análisis y Ranking	10
Introducir Resultados Arbitraje a Servidor	5
Generación Memoria Implementación	8
Total Implementación	170

Las tareas de la fase de pruebas, así como el tiempo invertido en cada una son los siguientes:

Tareas de Pruebas	Tiempo (Nº horas)
Prueba Ontología Genérica	5
Prueba Base de Reglas	8
Comprobación de resultados correctos	10
Prueba Generación de Historial	10
Prueba de Comunicaciones mediante MYSQL	5
Prueba Motor de Razonamiento Final	40
Generación Memoria Pruebas	67
Total Pruebas	145

Las tareas de la fase de integración, así como el tiempo invertido en cada una son los siguientes:

Tareas de Integración	Tiempo (Nº horas)
Ajustar comunicaciones Historial MySQL	8
Sustituir Motor de Razonamiento en Servidor	10
Ajustar comunicaciones resultados MySLQ	7
Sincronizar puertos del servidor	10
Integración Experimenter	4
Pruebas Experimenter	26
Generación Memoria Integración	10
Total Integración	75

6.3. Recursos

Mediante diagramas RBS representaremos la estructura tanto de los recursos humanos como materiales y tecnológicos.

Los recursos humanos estará compuestos por 3 personas. Consideramos necesario un jefe de proyecto, un analista y un programador. El analista es el encargado de desarrollar las tareas, diseñar el proyecto y dar apoyo al Jefe de Proyectos. El programador realiza la implementación de la aplicación y las pruebas unitarias correspondientes.

Analista y programador estarán dirigidos por un Jefe de proyecto, que será el responsable del proyecto y toma decisiones.

Se han representado los recursos humanos en la figura 64.

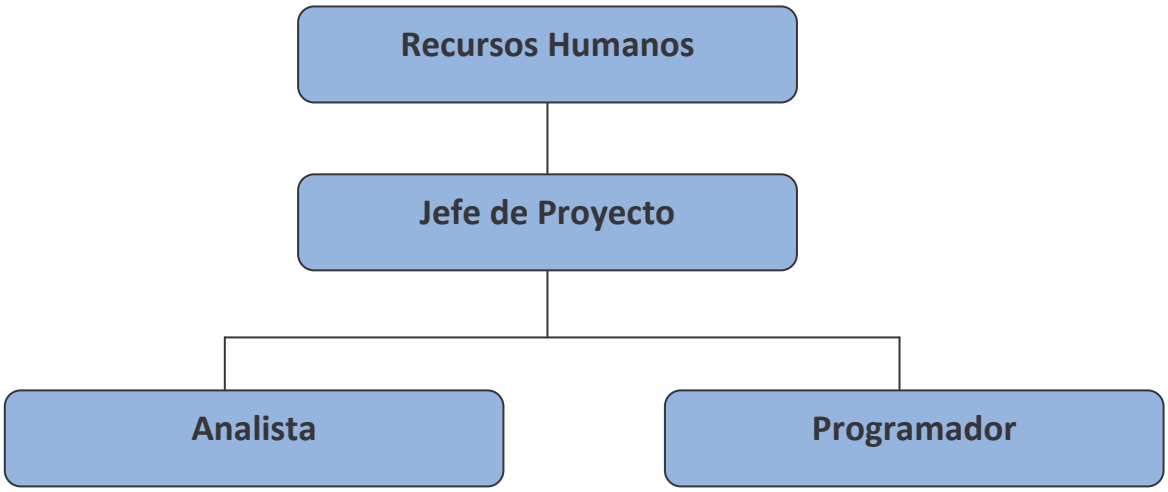


Figura 64: Organigrama de recursos humanos.

A pesar de que el proyecto real ha sido analizado, diseñado y desarrollado por una única persona, consideramos oportuno diferenciar estos tres roles para hacer más real el proyecto. En el caso del proyecto presente, la única persona implicada desarrollará funciones del Jefe de proyecto, analista y programador.

Los recursos materiales están compuestos por las herramientas, plataformas y demás recursos necesarios para la realización del proyecto.

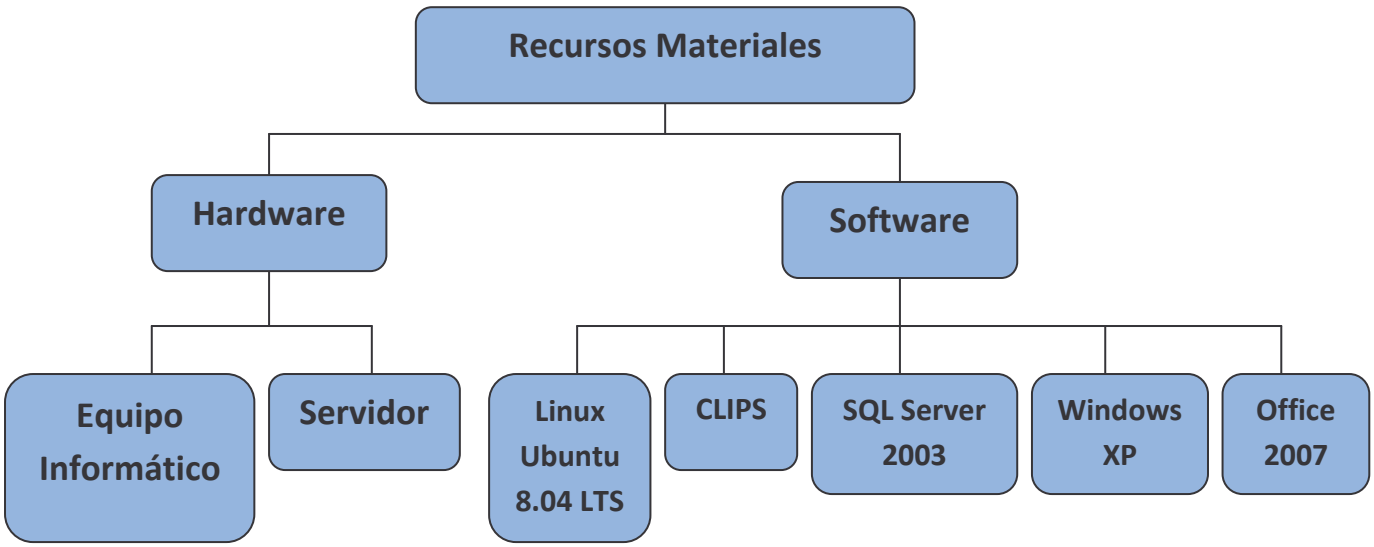


Figura 65: Organigrama de recursos materiales

6.4. Planificación del Proyecto

Las actividades que se van a realizar para llevar a cabo este proyecto se han descrito de acuerdo al estándar IEEE 1074-2006.

En la figura 66 se muestra el diagrama Gantt realizado con Microsoft Project 2007.

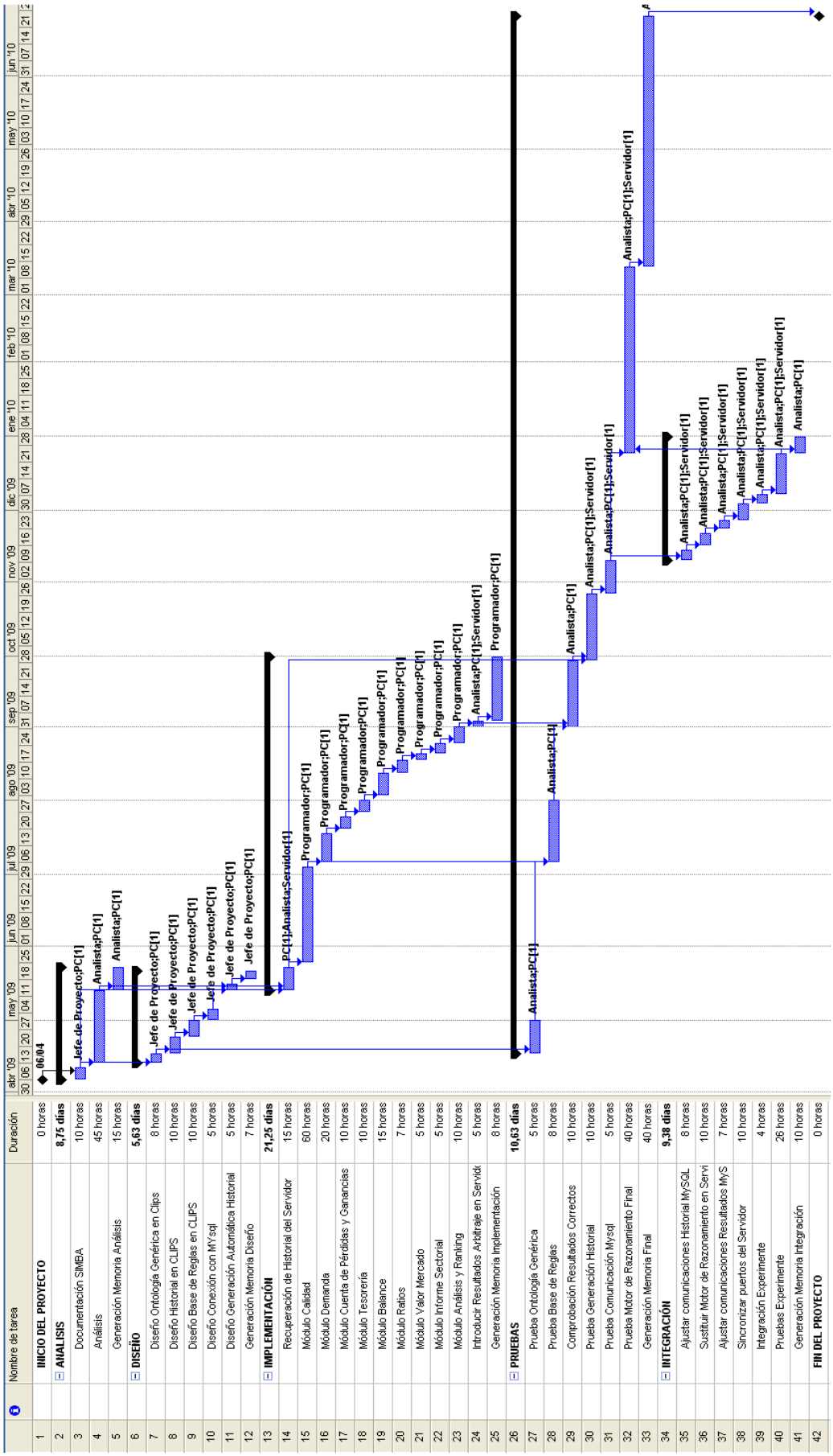


Figura 66: Diagrama de Gantt del proyecto

Después de la construcción de las técnicas de gestión de proyectos, realizando el diagrama de Gantt, hemos podido obtener que la duración total del proyecto es de 313 **días** laborales. Este proyecto no es un proyecto normal en cuanto a la asignación de recursos porque solamente hay una persona física como recurso humano, que desarrolla los 3 roles del proyecto. Además no ha habido dedicación exclusiva a la realización del proyecto de fin de carrera, sino que también se ha compatibilizado esta tarea con clases, prácticas, exámenes y más actividades académicas universitarias. Es por esta razón que los recursos humanos sólo trabajan a un 25% de sus posibilidades, lo que alarga la duración del proyecto. El proyecto empezó el día **6 de Abril de 2009** y terminó el día **17 de Junio de 2010**. El coste total del proyecto es de **20.145,00 €.**

A continuación se especifica un desglose de los costes directos, indirectos y de los beneficios previstos:

PRESUPUESTO DEL PROYECTO			
Tipo de Coste	Familia	Concepto	Coste
Costes Directos	Recursos Humanos	Jefe de Proyecto	2.200,00
		Analista	9.120,00
		Programador	4.800,00
Costes Indirectos	Hardware	Amortización de Equipo Informático	480,00
		Amortización de Servidor	330,00
		Coste Líneas de Teléfono e Internet	100,00
	Software	Amortización de Licencia Windows XP	50,00
		Amortización de Licencia Office 2007	25,00
		Amortización de Licencia SQL Server	40,00
	Proyecto	Reserva de Gestión	3.000,00
		PRESUPUESTO TOTAL:	**20.145,00**

La carga de trabajo de cada miembro del equipo en número de horas es la siguiente:

Cargo	Tiempo (Nº horas)	Coste (€/hora)	Total (€)
Jefe de Proyecto	55	40,00	2.200,00
Analista	240	38,00	9.120,00
Programador	210	22,86	4.800,00

Capítulo 7

Conclusiones

En este proyecto se ha realizado el diseño e implementación de un motor de razonamiento basado en reglas para el simulador empresarial SIMBA.

Como conclusión principal del desarrollo de este proyecto se extrae la puesta en práctica de conceptos clave en la formación de un ingeniero en informática adquiridos durante los estudios realizados. Entre estos conceptos, se puede destacar los relativos a la inteligencia artificial, ingeniería del software, manejo de bases de datos y gestión de proyectos. Es también destacable el beneficio de haber tenido que responder a la presión propia de los requerimientos inherentes al desarrollo de un sistema que se pretende implantar en un grupo de trabajo empresarial.

También es valiosa la experiencia a nivel de colaboración en grupos de trabajo, ya que para la consecución de los objetivos fijados, ha sido necesaria la colaboración de varias personas del laboratorio del grupo PLG. Ahondando en el tema de la colaboración, con respecto a los recursos empleados, es digno de mención el trabajo realizado para mantener los servidores del PLG y los equipos empleados.

Dado que este proyecto se ha llevado a cabo en los laboratorios de un grupo de investigación universitario que trabaja en proyectos reales para empresas, resulta necesario comentar lo enriquecedor de la experiencia de estar trabajando en un entorno de investigación, en el que se realizan multitud de proyectos. De esta forma, cada miembro del laboratorio aun estando asignado a un proyecto concreto, ha prestando ayuda siempre en las muchas ocasiones que ha sido necesario. En este caso en concreto, puesto que la temática abarcada por el presente proyecto implica al simulador empresarial SIMBA, ha sido necesaria la colaboración de miembros del laboratorio que crearon dicho sistema.

Es reseñable también la adaptación de los logros obtenidos en el desarrollo del proyecto al conjunto de objetivos que se proponían en el capítulo 3 de este documento, para alcanzar un mayor nivel de análisis, vamos a evaluarlos uno por uno:

- Se ha realizado un detallado estudio y análisis del sistema SIMBA a través de la documentación existente, lo cual posibilitó realizar un diseño bastante eficiente para mejorar el tiempo de arbitraje.

- Considero que he conseguido una familiarización con la tecnología utilizada en el proyecto, aprendiendo en algunos casos desde unas bases muy básicas que se tenían, como es el caso de MySQL Server. En otros casos este proyecto ha dado la posibilidad de profundizar y profesionalizar conocimientos que se tenían con anterioridad, como es el caso de los lenguajes de programación c++ y la herramienta CLIPS.

- Definitivamente se ha conseguido una implementación eficiente del motor de razonamiento basado en reglas, especificando los siguientes aspectos:

    - Un buen diseño de la ontología fue el mejor de los comienzos, ya que se consiguió una ontología fiel a la que existía con anterioridad, que a su vez incluye nuevos términos a fin de realizar una gestión eficiente.

    - El diseño e implementación del motor basado en reglas ha ocupado la mayor parte del tiempo y los esfuerzos requeridos por este proyecto, pero también ha causado la mayor satisfacción a nivel de ingeniería una vez terminado, al demostrar una mayor eficiencia con respecto a tiempos.

    - Una de las mejores ideas del proyecto tuvo mucho que ver con el acceso a la base de datos, dado que agilizó el motor de razonamiento. De esta manera tan sólo hace falta acceder al servidor antes y después del arbitraje. Trabajando durante el arbitraje con ficheros a nivel local, inspirado en como trabaja un sistema operativo con la memoria caché.

    - Se almacenan los resultados de cada arbitraje en el servidor mediante MySQL de manera que pueden ser consultados en cualquier momento desde un terminal con conexión a Internet.

    - Una de las dificultades finales estuvo en integrar el motor de razonamiento basado en reglas en el sistema SIMBA, pero finalmente con ayuda de creadores del mismo sistema, se logró realizar un buen trabajo con respecto a las comunicaciones internas del sistema para que quedase óptimamente integrado.

- Creo que se ha realizado un gran trabajo en el ámbito de la validación de resultados producidos por el gran número de simulaciones como se explicó en el Capítulo 5 de resultados. Además para la realización de este objetivo hay que destacar la herramienta *Experimenter*,cuyo manejo fue facilitado en gran medida por los miembros del laboratorio de investigación.

En lo referente a temas puramente técnicos, es de gran valor la experiencia adquirida en el conjunto de tecnologías de Business Intelligence. A nivel personal, considero este punto especialmente importante ya que este tipo de sistemas están siendo implantados en la mayoría de las empresas que pretenden asumir un perfil de modernidad en sus métodos de trabajo.

Capítulo 8

Futuras Líneas de Trabajo

Las futuras líneas de trabajo e investigación de este proyecto son abundantes por diferentes motivos como la nueva moda en la investigación en el E-Business o el aprendizaje por refuerzo. Sin embargo hay algunas posibilidades bastante factibles que cabe destacar:

Una línea futura de trabajo relacionada directamente con este proyecto podría ser la creación de una interfaz gráfica que facilitase el tratamiento de las reglas que marcan el entorno económico y financiero. De esta manera, cualquier usuario podría eliminar, modificar o añadir nuevas reglas que delimitasen o describiesen nuevos patrones de comportamiento en el entorno financiero. Esta interfaz gráfica sencilla pero eficiente, eliminaría la necesidad de que solamente un usuario especializado (con conocimientos de programación en sistemas de producción como CLIPS) pudiese modificar las reglas empresariales que rigen un arbitraje de simulación.

Con respecto a la investigación llevada a cabo por el departamento, un proyecto que se podría derivar de este estudio es el desarrollo de un agente inteligente basado en reglas que sea capaz de tomar decisiones automáticamente sin ningún tipo de intervención humana. Este sería un trabajo interesante debido a que permite la posibilidad de crear una simulación donde todas las empresas estén dirigidas por máquinas, haciendo que esta simulación evolucione de manera absolutamente independiente y automática a la vez que se arbitra un gran numero de periodos con rumbos financieros diferentes. Como resultado de este estudio se podría llegar a determinar las decisiones más adecuadas, según en que momento y entorno.

Este agente inteligente solamente recibiría como entrada unos algoritmos en lenguaje matemático para generar tomas de decisiones, y automáticamente crearía una base de reglas en formato CLIPS capaz de generar todos los periodos una toma de decisiones basándose en el historial y en los algoritmos introducidos. Además la base de reglas podría ser creada con la herramienta CLIPS, que como ya hemos visto tiene un alto grado de facilidad para cambiar los algoritmos que determinan la toma de decisiones. Si se quisieran cambiar los algoritmos matemáticos, únicamente habría que eliminar de la base de reglas las reglas antiguas y añadir las nuevas reglas que determinan los algoritmos matemáticos, sin necesidad de perder tiempo en parar la aplicación para compilar.

Bibliografía

[1] D. Borrajo, N. Juristo, V. Martínez y J. Pazos: "*Inteligencia Artificial. Métodos y Técnicas*". Centro de Estudios Universitarios Ramón Areces, Madrid, 1993.

[2] Alonso A, Guijarro B, Lozano A, Palma JT, Taboada MJ: "*Ingeniería del Conocimiento*". Aspectos Metodológicos Prentice Hall, 2004.

[3] Gómez A., Juristo N., Montes C., Pazos J.: "*Ingeniería del Conocimiento*". Centro de Estudios Ramón Areces, S.A.

[4] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García and Ismael Sagredo: "*SIMBA: A Simulator for Business Education and Research*". Decision Support Systems 48, pp 498-506. Elsevier 2010. *http://dx.doi.org/10.1016/j.dss.2009.06.009*

[5] Raquel Fuentetaja, Silvia de Castro, Javier García, Fernando Fernández, Daniel Borrajo y Fernando Borrajo:"*Un Simulador Empresarial como Herramienta Práctica para la Asignatura de Aprendizaje*". Jornadas de Enseñanza Universitaria de la Informática (JENUI 2010). Santiago de Compostela, España, 2010. <http://www.plg.inf.uc3m.es/%7Effernand/papers/jenui10.pdf>

[6] D.Borrajo, D.Pérez: PFC "*Generación automática de textos literarios: mitología griega*" Universidad Carlos III Madrid, 2004