Universidad
Carlos III de Madrid

Departamento de Informática

PROYECTO FIN DE CARRERA

SISTEMA WEB DE EDICIÓN DE REGLAS DE UN SIMULADOR EMPRESARIAL

Ingeniería Técnica en Informática de Gestión

Autor: Víctor Varillas Ledesma

Tutor: Fernando Fernández Rebollo

Leganés, 27 de Julio de 2012

Título: Sistema Web de Edición de Reglas de un Simulador Empresarial

Autor: Víctor Varillas Ledesma

Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 27 de Julio de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

RESUMEN

El principal objetivo de este proyecto consiste en el desarrollo de un sistema web para la edición y mantenimiento de un sistema de producción. El sistema de producción está basado en CLIPS. El proyecto se desglosa en las siguientes fases:

- Acceso a la meta información de la base de datos del simulador, para obtener la información sobre los elementos que se pueden utilizar en las reglas.

- Acceso a la base de reglas del sistema de producción, para cargar las reglas actuales.

- Desarrollo del interfaz web, con operaciones de borrado, edición y creación de reglas del sistema de producción.

Palabras clave:

Sistema – web – edición – reglas – simulador – empresarial - producción – CLIPS – SIMBA – negocio – experto – ontología – predicción – simulación

ABSTRACT

The main objective of this project is to develop a web system for editing and maintenance of a production system. The production system is based on CLIPS. The project is broken down into the following phases:

- Access to meta information of the database simulator to obtain information about items that can be used in the rules.

- Access to the rule base of the production system to load the current rules.

- Development of the web interface, with delete operations, editing and creating rules of the production system.

Keywords:

System – web – edition – rules – simulator – business – production – CLIPS – SIMBA – expert – ontology – predicction – simulation

ÍNDICE

ÍNDICE DE FIGURAS

TERMINOLOGÍA (ACRÓNIMOS Y DEFINICIONES)

ADA: Es un lenguaje multipropósito, orientado a objetos y concurrente, pudiendo llegar desde la facilidad de Pascal hasta la flexibilidad de C++.

Basic: Siglas de *Beginner's All-purpose Symbolic Instruction Code*, es una familia de lenguajes de programación de alto nivel. Fue originalmente desarrollado como una herramienta para la enseñanza y sigue siendo popular hasta el día de hoy.

BBDD: Acrónimo de bases de datos. Se define como un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

C: Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel.

CLIPS: Acrónimo de *C Language Integrated Production System* (sistema de producción integrado en Lenguaje C). Es una herramienta que provee un entorno de desarrollo para la producción y ejecución de sistemas expertos.

DBMS: Acrónimo de *Database Management System* (sistema gestor de bases de datos) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

DROOLS: Es un SGRN (sistema gestor de reglas de negocio) con un motor de reglas basado en inferencia de encadenamiento hacia delante, más correctamente conocido como sistema de reglas de producción, usando una implementación del algoritmo Rete.

DSL: Acrónimo de *Domain-Specific Language*. En desarrollo de software e ingeniería de dominio, un lenguaje específico del dominio es un lenguaje de programación o especificación de un lenguaje dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular. Un ejemplo de esto puede ser Maxima.

Fortran: Es un lenguaje de programación de alto nivel de propósito general, procedimental e imperativo, que está especialmente adaptado al cálculo numérico y a la computación científica.

IDE: *Integrated Development Environment*. Un entorno de desarrollo integrado es un programa informático compuesto por un conjunto de herramientas de programación destinado a crear aplicaciones. Puede dedicarse a un solo lenguaje de programación o bien puede utilizarse para varios.

Java: Es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel.

JavaDoc: Utilidad de Oracle para la generación de documentación de API's en formato HTML a partir de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java. La mayoría de los IDE's los generan automáticamente.

Java Servlet: Los servlet son objetos que corren dentro y fuera del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. Un servlet es un programa que se ejecuta en un servidor. El uso más común de los servlet es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

LALR parser: Look Ahead LR parser. Basado en un concepto de autómata finito. La estructura de información usada por un parser LALR es un autómata Pushdown (PDA). Un PDA determinista es un autómata finito determinista con la

adición de una pila de memoria indicando por qué estados se ha pasado hasta llegar al estado actual.

LISP: Es una familia de lenguajes de programación de computadora de tipo multiparadigma con una larga historia y una sintaxis completamente entre paréntesis. Fue creado originalmente como una notación matemática práctica para los programas de computadora.

LR parser: Son un tipo de analizadores para algunas gramáticas libres de contexto. Pertenece a la familia de analizadores ascendentes, ya que construyen el árbol sintáctico de las hojas hacia la raíz.

Máxima: Es un motor de cálculo simbólico escrito en lenguaje LISP publicado bajo licencia GNU GPL.

Parser: Un parser es un analizador sintáctico. Un analizador sintáctico es una de las partes de un compilador que transforma su entrada en un árbol de derivación. El análisis sintáctico convierte el texto de entrada en otras estructuras que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada.

POO: Programación Orientada a Objetos. Paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.

SGRN: Sistema Gestor de Reglas de Negocio. Software usado para definir, desplegar, ejecutar, monitorizar y mantener la variedad y complejidad de decisión lógica que es usada por sistemas operacionales dentro de una organización o empresa.

SIMBA: SImulator in Bussiness Administration. Es un simulador que emula la realidad empresarial utilizando las mismas variables, relaciones y eventos que se encuentran en el mundo de los negocios. Ha sido desarrollado por el Grupo de Investigación en Planificación y Aprendizaje Automático de la UC3M y por Simuladores Empresariales S.L., una spin-off tecnológica creada por el Grupo de Investigación INNOVATIC, del Departamento de Organización de Empresas de la Universidad Autónoma de Madrid.

Spin-Off: Proyecto nacido como extensión de otro anterior, o más aún de una empresa nacida a partir de otra mediante la separación de una división subsidiaria o departamento de la empresa para convertirse en una empresa por si misma.

1. INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL DOCUMENTO

El documento que tiene entre sus manos forma parte del Proyecto Fin de Carrera **"Sistema Web de Edición de Reglas de un Simulador Empresarial".** Este proyecto es una parte de otro proyecto global que tiene como objetivo el crear un editor de reglas que permita añadir, modificar y eliminar las reglas que componen el motor de razonamiento de un simulador empresarial.

Luis Echevarría Armero ha sido el autor de la otra parte de este proyecto global. Su proyecto está dedicado principalmente al desarrollo de la parte visible del editor, la interfaz de uso, el control de usuarios, etc.

Este proyecto está enfocado a la **verificación y traducción** de las reglas, el flujo de ejecución de la simulación, etc., aún así la colaboración entre ambos ha sido muy estrecha y el *feedback* continuo ya que **ambas partes están íntimamente ligadas**. Sin esa continua comunicación hubiera sido imposible llevar el proyecto a buen puerto. Más adelante se explicará con más detalle el propósito de la solución aportada, pero antes es necesario conocer otros aspectos del entorno donde nos moveremos.

Se comienza dando una pequeña introducción hablando sobre los **simuladores empresariales**, para qué se usan, qué implicaciones tienen, etc. y se tratan también otros sistemas basados en la inteligencia artificial que tienen relación con estos simuladores.

Posteriormente se continua con el proyecto en sí, la posición de partida, cual es la problemática original que se desea resolver con la solución, qué análisis se ha efectuado del problema para realizar el diseño, la implementación de la solución y la integración con el resto de componentes.

Se continúa con la validación de la solución diseñada, se comprueba si verdaderamente resuelve el problema original en unos tiempos razonables y si es útil y manejable esta solución para el usuario. También se verificará si los objetivos de integridad y robustez se ven cumplidos o no.

En la parte final del documento se analiza qué posibles mejoras se pueden acometer en la solución aportada, como añadir nuevas funcionalidades o flexibilizar la forma en la que se crean cálculos y reglas nuevas. Esta parte final irá acompañada de una sección donde se detalla la gestión del proyecto describiendo los recursos económicos necesarios y la planificación que se llevó a cabo para el desarrollo satisfactorio del mismo. Por último en la sección de apéndices se adjuntan otra serie de documentos considerados menos relevantes pero útiles para formar parte de la documentación final.

En las secciones que vienen justo a continuación se procede con una descripción del contexto donde se desarrolla el proyecto para tener una mejor visión y entendimiento de lo que se plantea hacer.

1.2 DESCRIPCIÓN DEL CONTEXTO

En esta sección que aquí comienza se listan y especifican diferentes ideas, conceptos y tecnologías que son necesarias entender para poder comprender de mejor forma el objetivo y solución del proyecto. Se comienza hablando sobre los **simuladores empresariales** y más concretamente de **SIMBA** (*Simulator In Business Administration*). Se continua con los **sistemas basados en reglas**, los cuáles permiten construir este tipo de simuladores, y **CLIPS**, que es la tecnología usada en este proyecto. Por último, se describe qué son los **sistemas de gestión de reglas de negocio**, usados para mantener la variedad y complejidad de decisión lógica dentro de una empresa y se ejemplifica con **DROOLS**, una de las tecnologías más usadas. Empecemos con los simuladores empresariales.

1.2.1 SIMULADORES EMPRESARIALES

Día a día, la cantidad de directivos y profesionales de todo tipo preocupados por su formación aumenta considerablemente. Los cargos y responsabilidades que estos desempeñan son cada vez más variados y complejos a causa de la rápida evolución del mercado empresarial, el cuál ofrece amplias y múltiples posibilidades a aquellas personas con una **sólida formación y una visión real de los negocios.** [http://e-archivo.uc3m.es/handle/10016/6825]

Existen diversas formas de afrontar el reto de la formación según los recursos disponibles (tanto en tiempo como económicos) así como de la oferta de cursos y programas al alcance del interesado. La mayor parte de estos cursos requieren **largas formaciones presenciales** y un estudio de un material escrito que suele ser **bastante voluminoso**. Frente a esta situación el interesado tiende a adoptar una **actitud pasiva** ante el estudio, donde él no es el jugador principal en la formación.

Otra alternativa a esta formación es aquella relacionada con el uso de **programas de simulación**. En esta metodología destaca el rol que adquiere el participante implicándose en el proceso de formación al integrarse en el equipo de dirección de una empresa compitiendo con otras compañías en un mercado simulado en el ordenador.

De esta manera se logran desarrollar una serie de habilidades y alcanzar unos objetivos que de otra manera no serían posible, como un **mayor desarrollo de la capacidad de trabajo en equipo, rapidez y solvencia** a la hora de tomar decisiones sin asumir los riesgos reales de negocio, obtención inmediata de los resultados según las decisiones tomadas, etc.

Todas estas razones hacen más que justificable el uso de estas herramientas tanto para la formación inicial en escuelas de Administración y dirección de empresas, como para programas más específicos e intensivos orientados a gerentes y directivos.

El área de conocimiento de los simuladores empresariales cubre el diseño y desarrollo de software, procesos y las mejores prácticas para **integrar,**

almacenar y analizar la información del mercado. Identificar el área correcta a cambiar es un factor importante a la hora de mejorar y dirigir una empresa hacia el éxito.

Estas herramientas son muy útiles para ayudar a los directivos de las empresas a entender los **procesos de negocio** y cómo la modificación de estos procesos impacta en la organización. De esta manera, el riesgo que conlleva estos cambios es identificado antes de implementar las decisiones. Una vez que los factores de riesgo han sido identificados el simulador empresarial puede ser usado para cambiar los parámetros deseados.

Los simuladores empresariales son también herramientas muy prometedoras para la **investigación y la enseñanza**. Las simulaciones emulan los problemas del mundo real; el éxito o fracaso de una organización depende de cómo los dirigentes toman decisiones en cuanto al precio, marketing y capital para inversión. Sin embargo, también ofrecen un amplio abanico de posibilidades para otro tipo de estudios como herramienta para dirigir investigaciones psicológicas. En este caso, el simulador puede ser usado para investigar el proceso de toma de decisiones en situaciones complejas y obtener conclusiones sobre la mente humana y su habilidad para resolver aparentemente situaciones caóticas. Usos típicos de los simuladores en la investigación son: **planificación financiera** (cuantificando el impacto de las decisiones tomadas), **gestión del riesgo** (medida, gestión y determinación del equilibrio perfecto entre beneficio y riesgo), **predicción** (análisis de información obtenida anteriormente por el simulador para predecir el futuro), **aprendizaje interactivo** (los simuladores pueden ser usados para lecciones de economía) e investigación de la **inteligencia artificial** (minería de datos, agentes inteligentes, etc.).

En cuanto al estado del arte en los simuladores empresariales, decir que, por supuesto, los avances en las tecnologías de la información han mejorado la experiencia de juego, proveyendo una respuesta más rápida y una mejor usabilidad y accesibilidad. Los resultados en educación demuestran el éxito de estos productos software. Las simulaciones empresariales mejoran la velocidad de transferencia de la teoría a la práctica con costes reducidos y ahorros en el tiempo de entrenamiento. Representan un cambio en los modelos mentales, haciendo conexiones más rápidas entre la percepción de las decisiones y las acciones mostrando de forma sobrada su efectividad a la hora de lograr un cambio en los procesos mentales involucrados en las decisiones. Como resultado, la mejora en las capacidades, competencias, habilidades, etc. está más que demostrada. **SIMBA** representa un gran ejemplo del uso de los simuladores empresariales como herramienta educativa. Se explica en la siguiente sección.

1.2.2 SIMBA

SIMBA (SIMulator in Business Administration) es el trabajo de una *spin off* creada en la **Universidad Autónoma de Madrid** en colaboración con la **Universidad Carlos III de Madrid**. SIMBA es un programa basado en web que simula el rendimiento de una serie de mercados. Este programa es el resultado de 20 años de experiencia en simuladores empresariales tanto en educación universitaria como en entrenamiento para ejecutivos. En este se emula la realidad de los negocios usando las mismas variables, relaciones y eventos presentes en el mundo empresarial. El propósito es proveer a los usuarios de una visión integrada de la compañía, usando las mismas reglas básicas, relaciones y dinámicas del mercado presentes en la gestión de los negocios, simplificando la complejidad y destacando el contenido. [http://e-archivo.uc3m.es/handle/10016/6825]

SIMBA tiene un número de características que valen la pena destacar. Primero, es un **simulador de competición**, donde un equipo de participantes pueden competir frente a otras compañías manejadas automáticamente por el simulador a través de agentes inteligentes, o donde el resto de empresas pueden estar dirigidas por otros participantes. También es multifuncional porque se dirige a las principales áreas funcionales de la empresa. Es interactivo porque permite a los participantes comunicarse con el simulador y con otros participantes.

Es muy versátil porque provee diferentes niveles de dificultad, adaptándose al conocimiento de los participantes en administración de empresas. No hay límite en el número de usuarios. Permite la diversificación de productos, mercados y tecnologías. Como es un sistema web los usuarios pueden acceder desde cualquier ordenador conectado a Internet, eliminando así la necesidad de instalación de la aplicación en un entorno local. Además puede ser usado en cualquier plataforma y sistema operativo, en cualquier lugar y dispositivo con acceso a la web. **SIMBA** puede ser adaptado a la mayoría de requisitos de entrenamiento, usando distintos lenguajes y dialectos, monedas y factores socioeconómicos. Características adicionales son la existencia de un amplio número de gráficos y hojas de cálculo como indicadores principales para el seguimiento de nuestra compañía y del mercado así como un módulo de ranking y evaluación para valorar objetivamente el rendimiento de la gestión del equipo. Todas estas características representan una herramienta única para crear una

inmersión efectiva en el entorno de aprendizaje para alcanzar los objetivos educacionales.

En cuanto a la arquitectura de **SIMBA** decir que este está compuesto de un conjunto de subsistemas especializados diseñados para proveer altos estándares de versatilidad y usabilidad, tanto como para el participante como para el administrador de la simulación. Entre los módulos del simulador nos encontramos con el módulo de usuarios, el cual permite al administrador del sistema definir los 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 para ejecutarse automáticamente a la hora de responder a ciertos eventos predefinidos sin requerir la participación del administrador.

El módulo de **Adaptación** (*Customization Module*) cubre un rango de funciones destinadas a crear productos, entornos geográficos y económicos, situaciones iniciales de las compañías, novedades para el boletín de noticias del mercado, junto con la selección de moneda y el idioma de la competición. Toda esta información permite la generación de simulaciones hechas a medida de las necesidades de los participantes. Este módulo provee una capacidad adaptativa interesante e innovativa y es la piedra angular para explotar el potencial de **SIMBA**.

El módulo de **Planificación** permite al instructor organizar y definir las características de cada simulación. Puede crear equipos asignándoles participantes, determinando el número de decisiones a ser tomadas y su planificación. Definiendo el número de mercados y el número de empresas en cada mercado y asignando equipos a esas empresas. Posteriormente, en el módulo se deben definir cómo los mercados serán estructurados en términos de moneda, idioma, producto y otros parámetros económicos y de entorno. El módulo de **Conocimiento** (*Knowledge Module*) está creado para proveer soporte online y guías a los participantes.

El proceso de simulación consta de cuatro módulos especializados que configuran la dinámica de competición que es la que realmente los usuarios perciben cuando participan en la simulación. Cada uno de los módulos realiza una tarea específica. El proceso comienza con el módulo de **Informes** (*Reporting Module*), el cual genera toda la información que el equipo necesita sobre su empresa, sus competidores y el entorno de competición.

En los módulos de **Análisis y Diagnosis** los participantes del equipo aplican sus habilidades para diagnosticar la posición de competitividad de la empresa y definir objetivos estratégicos y operacionales. Finalmente, se toman las decisiones apropiadas usando la interfaz **SIMBA** en el módulo de **Decisión** (*Decision Making Module*), incluyendo negociaciones de *outsourcing*. Estos módulos forman el escenario central en el cual los participantes toman sus roles como jugadores. Hacen uso de aproximadamente **25 variables** por mercado, organizadas en áreas funcionales, una cifra que puede ser aumentada si las compañías entablan negociaciones de *outsourcing*. El tiempo medio de toma de decisiones es de dos horas por mercado.

El módulo de **Arbitraje** (*Arbitrage Module*) está pensado para controlar la dinámica de la competición y la interacción entre equipos e instructores. Este es ejecutado cada vez que un período de toma de decisiones termina, de acuerdo al calendario planificado, verificando que todos los equipos han tomado sus decisiones. Este módulo también permite al instructor seleccionar noticias e incidentes, algunos de los cuales pueden ser creados en el **módulo de Adaptación** (*Customization Module*), que pueden alterar la dinámica del mercado, adaptando así el simulador a las habilidades de los participantes. Después el sistema de arbitraje del mercado es puesto en marcha. Haciendo esto, el programa integra las situaciones anteriores del simulador con las decisiones de los equipos, con los parámetros del entorno económico y con cada uno de los factores geográficos del mercado. **Después el motor de simulación comienza a producir información para el nuevo período.** Este proceso es llevado a cabo por todas las compañías en un mercado y después por todos los mercados que compiten en **SIMBA**. Una vez finalizado el ciclo los resultados estarán disponibles para todos los participantes para comenzar un nuevo ciclo de decisión hasta el final de la competición que nos llevará al módulo **Ranking y Evaluación**.

Al finalizar, **SIMBA** ofrece una clasificación de los equipos participantes en cada simulación. Para realizar esta evaluación se usa un procedimiento de

múltiples criterios en el cual el comportamiento de una serie de indicadores (principalmente económicos, comerciales, financieros, etc.) son analizados. **SIMBA** también permite al instructor dar un peso o importancia a cada uno de estos indicadores dependiendo de la importancia que se quiera dar a cada uno de ellos. Aún más, cuando hay múltiples mercados en una simulación, un sistema más complejo de evaluación puede ser aplicado.

Al final de cada etapa de toma de decisiones se produce un proceso que lanza los eventos relacionados con el entorno económico, dinámicas de mercado y procedimientos administrativos que generan una nueva situación de mercado, que inician un nuevo período de toma de decisiones. Este proceso es "lanzado" por el instructor y tiene lugar para cada uno de los mercados que componen la simulación. El arbitraje comienza cuando el proceso de toma de decisiones expira, esto significa que todos los equipos deben haber tomado sus decisiones para cada uno de los mercados en los que operan. El instructor comprueba que todos los equipos acabaron, interviene en el entorno si decide realizar algún cambio significativo en las condiciones de mercado y finalmente lanza el motor de simulación para obtener una nueva situación.

El primer paso en el arbitraje es definir el contexto económico. Las variables que intervienen son, entre otras, la inflación, el índice de confianza del consumidor, intereses tanto para préstamos como para activos financieros, precios, etc. Estas se comportan de una manera aleatoria que comienza con la situación especificada por el instructor y evolucionan según dicta la inflación. El arbitraje de demanda del mercado (*Market Demand Arbitrage*) está pensado para generar la demanda total de productos que el mercado desea absorber bajo las condiciones económicas del entorno y el esfuerzo de marketing global que las compañías hacen para estimular la demanda. La demanda está obviamente influenciada por las condiciones económicas pasadas y determinará el futuro comportamiento de la evolución del mercado. Las variables más significantes que afectan la demanda potencial son las proyecciones de demanda a largo plazo, estacionalidad de las ventas, esfuerzo de marketing de todas las compañías y otros parámetros de entorno obtenidos durante el periodo de arbitraje del entorno económico.

La evolución de la tecnología de la información y las comunicaciones, junto con la nueva demanda de modelos superiores de educación (adaptación a diferentes perfiles de usuario, aprendizaje a distancia, etc.) justifican el desarrollo

de nuevas herramientas de entrenamiento, como es el caso de **SIMBA**. Centrándonos más específicamente en las ventajas pedagógicas de los simuladores empresariales debemos destacar el aprendizaje por inmersión como una característica de esta metodología. Esto significa que los estudiantes usan técnicas y conceptos adquiridos durante su entrenamiento, y es necesario analizar, tomar decisiones y evaluar los resultados obteniendo así una experiencia práctica y un fuerte conocimiento. Está característica está mejorada por la posibilidad de escoger agentes inteligentes específicos para actuar como competidores, adaptando así el mercado y el comportamiento de los competidores a las características del grupo estudiante.

Hasta ahora, **SIMBA** ha sido usado en escuelas de administración y dirección de empresas y otros programas de entrenamiento en los cuales la administración de una empresa forma parte del contenido principal o del complementario. En esos contextos **SIMBA** ha jugado dos roles alternativos. Por un lado es una actividad integradora para los participantes proveyendo un entorno de negocios neutral en el cual interactúan, se conocen los unos a los otros y toman diferentes roles profesionales, y, por otro lado, es una herramienta para relacionar conceptos y poner en práctica el conocimiento y las técnicas enseñadas en otras materias.

En adición, hay más beneficios derivados de usar simuladores en la educación para los negocios. Algunos de ellos están relacionados con objetivos técnicos de aprendizaje, mientras otros están ligados con el desarrollo de capacidades y competencias altamente valiosas en sistemas modernos de educación. Si se necesita más información sobre este tema o ahondar más en profundidad en algún aspecto en concreto puede dirigirse a **http://e-archivo.uc3m.es/handle/10016/6825** . Este documento contiene la información aquí presentada expuesta de una forma más ampliada.

1.2.3 SISTEMAS BASADOS EN REGLAS

Los sistemas basados en reglas trabajan mediante la aplicación de una serie de reglas, comparación de resultados y aplicación de las nuevas reglas basadas en la nueva situación creada. Estos sistemas son uno de los modelos de representación del conocimiento más ampliamente utilizados debido a que resultan muy apropiados en situaciones en las que el conocimiento que se desea representar surge de forma natural con estructura de reglas. [http://personales.unican.es/gutierjm/cursos/expertos/Reglas.pdf]

Uno de los paradigmas de programación tradicionalmente asociado a los sistemas basados en reglas es el de los **sistemas de producción**. Desde el punto de vista semántico, hay dos aproximaciones principales a los lenguajes basados en reglas. El paradigma de la **programación lógica** (usado en bases de datos deductivas) y el paradigma de los **sistemas de producción**, que proporciona una semántica procedural basada en el encadenamiento hacia delante (o atrás) de reglas. Este último es el que se usa en los sistemas de producción típicos de los sistemas expertos. Estos sistemas se usan cuando se trabaja con un dominio en el que se comprende bien la teoría y donde el conocimiento se puede representar mediante hechos y reglas.

Las reglas deterministas constituyen la más sencilla de las metodologías utilizadas en sistemas expertos. La **base de conocimiento** contiene el conjunto de reglas que definen el problema, y el **motor de inferencia** saca las conclusiones aplicando la lógica clásica a estas reglas. Es por eso que las tres partes fundamentales de un sistema basado en reglas son: **la base de hechos, la base de reglas de producción y el motor de inferencia** (máquina deductiva).

ARQUITECTURA

En los sistemas basados en reglas intervienen dos elementos importantes: la **base de conocimiento y los datos**. Los datos están formados por la evidencia o hechos conocidos en una situación particular. Este elemento puede cambiar de una aplicación a otra, por esa razón no es de naturaleza permanente. En situaciones deterministas, las relaciones entre un conjunto de objetos pueden ser representadas mediante un conjunto de reglas. El conocimiento se almacena en la base de conocimiento y consiste en un conjunto de objetos y un conjunto de

reglas que gobiernan las relaciones entre esos objetos. La información almacenada en la base de conocimiento es de naturaleza permanente y estática, no cambia de una aplicación a otra, a menos que se incorporen elementos de aprendizaje.

Las reglas son afirmaciones lógicas que relacionan dos o más objetos e incluye dos partes, la **premisa y la conclusión**. La premisa es aquello que expresa la condición y está ubicada entre las palabras clave *si* y *entonces*. La premisa puede contener una o más afirmaciones. La conclusión es la expresión lógica que nos dictamina la acción a tomar para dar un nuevo valor a determinado objeto. Está situada detrás de la palabra clave *entonces*.

El **motor de inferencia** es la herramienta que permite obtener nuevas conclusiones y hechos a partir de los datos ya existentes y del conjunto de reglas almacenados en la base de conocimiento. Si la premisa de una regla es cierta, entonces la conclusión de la regla debe ser cierta también. Los datos iniciales se incrementan incorporando las nuevas conclusiones. Por ello, tanto los hechos iniciales o datos de partida como las conclusiones derivadas de ellos forman parte de los hechos o datos de que se dispone en un instante dado. Las conclusiones resultantes pueden clasificarse en simples o compuestas dependiendo de si se han obtenido mediante la aplicación de una sola regla o varias. Los expertos utilizan diferentes tipos de reglas y estrategias de inferencia y control para obtener nuevo conocimiento a partir del existente. Entre estas estrategias están la **deducción, inducción y abducción.**

El método de **deducción** se basa en que el nuevo conocimiento será cierto si se parte de conocimiento cierto. Esta es la fuerza de la inferencia lógica. El mecanismo de **inducción** es el del aprendizaje automático aunque presenta el problema de la dudosa fiabilidad del conocimiento inferido. El método de **abducción** se basa en que un conjunto de reglas y hechos observados produce un conjunto de explicaciones posibles que, usando la deducción, harían coherente el conocimiento de partida.

INFERENCIA EN SISTEMAS DE REGLAS

Las reglas de inferencia (*Modus Ponens, Modus Tollens* y Resolución) y las estrategias de inferencia (encadenamiento de reglas, encadenamiento de reglas orientado a un objetivo y compilación de reglas) son las utilizadas por el motor de inferencia para obtener conclusiones simples y compuestas.

El ***Modus Ponens*** es quizás la regla de inferencia más conocida ya que se utiliza para obtener conclusiones simples. En ella se analiza la premisa de la regla y si es cierta la conclusión pasa a formar parte del conocimiento. La regla ***Modus Tollens*** también se utiliza para obtener conclusiones simples. En este caso se examina la conclusión y si es falsa, se concluye que la premisa también es falsa. Ambas reglas son complementarias.

Las reglas de inferencia *Modus Ponens* y *Modus Tollens* pueden ser utilizadas para obtener conclusiones simples. Por otra parte, las conclusiones compuestas (aquellas basadas en dos o más reglas) se obtienen usando el llamado mecanismo de resolución que consta de tres etapas. En la primera las reglas son sustituidas por expresiones lógicas equivalentes, estas expresiones lógicas se combinan en otra nueva y finalmente se utiliza esta última expresión para obtener la conclusión. Estas etapas involucran conceptos tales como la combinación y la simplificación de expresiones lógicas, pero no es necesario entrar más en detalle.

Una de las estrategias de inferencia más utilizadas para obtener conclusiones compuestas es el llamado **encadenamiento de reglas**. Esta estrategia puede utilizarse cuando las premisas de ciertas reglas coinciden con las conclusiones de otras. Cuando las reglas son encadenadas los hechos pueden utilizarse para dar lugar a nuevos hechos. Esta dinámica se repite continuamente hasta que no es posible obtener más conclusiones. El tiempo que consume este proceso hasta su terminación depende de los hechos conocidos y de la cantidad de reglas que se activan.

El algoritmo de encadenamiento de reglas orientado a un objetivo requiere del usuario seleccionar, en primer lugar, una variable o nodo objetivo; entonces el algoritmo navega a través de las reglas en búsqueda de una conclusión para el nodo objetivo. Si no se obtiene ninguna conclusión con la información existente, entonces el algoritmo fuerza a preguntar al usuario en busca de nueva información sobre los elementos que son relevantes para obtener información sobre el objetivo.

Otra forma de tratar con reglas encadenadas consiste en comenzar con un conjunto de datos (información) y tratar de alcanzar algunos objetivos. Esto se conoce con el nombre de compilación de reglas. Cuando ambos, datos y objetivos, se han determinado previamente, las reglas pueden ser compiladas, es decir, pueden escribirse los objetivos en función de los datos para obtener las llamadas ecuaciones objetivo.

CONTROL DE COHERENCIA

En situaciones complejas, incluso verdaderos expertos pueden dar información inconsistente (reglas inconsistentes, combinaciones no factibles de hechos, etc.). Por ello, es muy importante controlar la **coherencia del conocimiento** tanto durante la construcción de la base de conocimiento como durante los procesos de adquisición de datos y razonamiento. Si la base de conocimiento contiene información inconsistente, es muy probable que el sistema experto se comporte de forma poco satisfactoria y obtenga conclusiones absurdas.

El objetivo del control de coherencia consiste en ayudar al usuario a no dar hechos inconsistentes (por ejemplo dándole al usuario las restricciones que debe satisfacer la información demandada), a evitar que entre en la base de conocimiento cualquier tipo de información inconsistente o contradictoria. El control de coherencia debe hacerse controlando la coherencia de las reglas y de los hechos.

EXPLICANDO CONCLUSIONES

Las conclusiones no bastan para satisfacer al usuario de un sistema experto. Los usuarios esperan también que el sistema les dé algún tipo de explicación que indique el por qué de las conclusiones. Durante el proceso realizado por el motor de inferencia, las reglas activas (las que han concluido) forman la base del mecanismo de explicación, que es regulado por el subsistema de explicación.

En los sistemas expertos basados en reglas, es fácil dar explicaciones de las conclusiones obtenidas. El motor de inferencia obtiene conclusiones basándose en un conjunto de reglas y, por tanto, conoce de qué regla procede cada conclusión. Por ello, el sistema puede dar al usuario la lista de hechos concluidos junto con las reglas que se han utilizado para obtenerlos.

APLICACIONES DE LOS SISTEMAS BASADOS EN REGLAS

Los sistemas basados en reglas, dada su naturaleza, son usados como sistemas de propósito específico para realizar tareas de soporte a expertos. La clasificación de las tareas que se clasifican en:

- **Clasificación**: Un sistema clasificador proporciona como salida una respuesta, seleccionada entre un conjunto de respuestas prefijado, conforme a un conjunto de entradas denominado situación.

- **Predicción y pronóstico:** Los sistemas de predicción analizan secuencias de datos distribuidas a lo largo del tiempo para tratar de prever situaciones venideras. Son aplicados a todos los dominios sujetos al cambio provocado por el paso del tiempo, como la predicción meteorológica, el pronóstico de valores inmobiliarios, etc.

- **Diseño:** Un sistema basado en reglas enfocado al diseño de un sistema es capaz de establecer la configuración del mismo atendiendo a sus requisitos, por ejemplo, minimizar una función objetivo que mide los costes de producción de un diseño.

- **Planificación:** La planificación consiste en la elaboración de una serie de acciones a seguir para alcanzar un objetivo.

- **Monitorización:** Los sistemas de monitorización observan un conjunto de variables de estado correspondientes a un sistema con el fin de alertar ante situaciones anómalas. Un requisito indispensable de los sistemas de monitorización es su funcionamiento constante, ya que su misión es la de vigilar constantemente al sistema monitorizado.

- **Ayuda inteligente:** Este tipo de sistema está enfocado a atender las necesidades del usuario de un sistema para proporcionarle consejo e información e información de interés. Los sistemas de ayuda a la decisión se usan como asesoramiento de expertos en diversos dominios.

- **Instrucción:** La finalidad de estos sistemas es instruir a un alumno en una determinada materia, localizando sus errores y ofreciéndole el conocimiento para subsanarlos.

CLIPS es una tecnología que permite modelar sistemas basados en reglas. Se habla de él a continuación.

1.2.4 CLIPS

CLIPS es una herramienta de sistemas expertos originalmente creada en el **Lyndon B. Johnson Space Center de la NASA**. Desde su primera aparición en 1986, CLIPS ha experimentado continuos refinamientos y mejoras. CLIPS es un acrónimo de *C **Language Integrated Production System*** (Sistema de Producción Integrado en Lenguaje C). CLIPS probablemente es el sistema de producción más ampliamente usado debido a que es rápido, eficiente y gratuito. Está diseñado para facilitar el desarrollo de software y adaptarlo al modelo de conocimiento humano. Existen tres formas de representar el conocimiento en CLIPS: **Reglas** (desarrolladas para representar el conocimiento heurístico basado en la experiencia), **funciones genéricas** (diseñadas para el conocimiento procedural), **programación orientada a objetos** (también usado para representar el conocimiento procedural). Las cinco características generalmente aceptadas de la programación orientada a objetos son soportadas en CLIPS: clases, manejadores de mensajes, abstracción, encapsulación, herencia y polimorfismo. Las reglas deben emparejarse con los objetos y los hechos. [http://es.wikipedia.org/wiki/CLIPS]

CLIPS también ha sido diseñado para la integración con otros lenguajes como C y Java. También puede ser usado como una herramienta autónoma. CLIPS puede ser llamado desde un lenguaje procedural, realizar su función y después devolver el control al programa que lo llama. Igualmente, el código procedural puede ser definido como una función externa y llamado desde CLIPS. Cuando el código externo completa su ejecución el control vuelve a CLIPS.

Como se mencionaba anteriormente, CLIPS es un tipo de lenguaje para escribir **sistemas expertos,** sistemas para modelar el conocimiento humano. CLIPS es llamada una herramienta para construir sistemas expertos porque ofrece un completo entorno de desarrollo para sistemas expertos incluyendo características tales como un editor integrado y una herramienta de depuración. La consola está reservada para aquella parte de CLIPS que realiza la inferencia o razonamiento. La consola CLIPS nos provee de los elementos básicos de un sistema experto: la **lista de hechos** (instancias), la **base de conocimiento** (que contiene todas las reglas) y el **motor de inferencia** (que controla la ejecución de las reglas). Un programa en CLIPS consta de las reglas, los hechos y los objetos. El motor de inferencia decide qué reglas deben ser ejecutadas y cuándo. Un sistema experto basado en reglas escrito en CLIPS es un programa dirigido por la

información donde los hechos y objetos son la información que estimula la ejecución del motor de inferencia. Este es un ejemplo de cómo CLIPS difiere de otros lenguajes procedurales como JAVA, Ada, BASIC, FORTRAN y C. En los lenguajes procedurales la ejecución puede continuar sin información.

Originalmente CLIPS sólo tenía capacidades para representar reglas y hechos, sin embargo, mejoras posteriores permitieron a las reglas relacionarse con los objetos al igual que con los hechos. También, los objetos pueden ser usados sin reglas, mandándose mensajes, de tal manera que el motor de inferencia no sería necesario si sólo se usasen objetos.

Resumiendo, las características principales de CLIPS son:

- **Representación del conocimiento:** CLIPS permite manejar una amplia variedad de conocimiento, soportando tres paradigmas de programación, el declarativo, el imperativo y el orientado a objetos. La programación lógica basada en reglas permite que el conocimiento sea representado como reglas heurísticas que especifican las acciones a ser ejecutadas dada una situación. La POO permite modelar sistemas complejos como componentes modulares.
- **Portabilidad:** CLIPS fue escrito en C con el fin de hacerlo portable y rápido y puede ser instalado en diversos sistemas operativos (Windows, MacOS X, Unix) sin ser necesario modificar si código fuente. CLIPS puede ser ejecutado en cualquier sistema con un compilador ANSI de C o un compilador de C++.
- **Integrabilidad:** CLIPS puede ser embebido en código imperativo, invocado como una sub-rutina, e integrado con lenguajes como C, Java, FORTRAN y otros. CLIPS incorpora un completo lenguaje orientado a objetos (COOL) para la elaboración de sistemas expertos. Aunque está escrito en C, su interfaz más próxima se parece a LISP.
- **Desarrollo interactivo:** La versión estándar de CLIPS provee un ambiente de desarrollo interactivo y basado en texto; este incluye herramientas para la depuración, ayuda en línea y un editor integrado.
- **Verificación / Validación:** CLIPS contiene funcionalidades que permite verificar las reglas incluidas en el sistema experto que está siendo desarrollado, incluyendo diseño modular y particionamiento de la base de conocimientos del sistema, chequeo de restricciones estático y dinámico para funciones y algunos tipos de datos, y análisis semántico de reglas para prevenir posibles inconsistencias.
- **Bajo costo:** CLIPS es un software de dominio público.

1.2.5 SISTEMAS DE GESTIÓN DE REGLAS DE NEGOCIO (SGRN)

Un sistema de gestión de reglas de negocio es un software de sistema usado para **definir, desplegar, ejecutar, monitorizar y mantener** la variedad y complejidad de decisión lógica que es usada por los sistemas operacionales dentro de una organización o empresa. Esta lógica, también referida como las reglas de negocio, incluye políticas, requisitos y sentencias condicionales que son usadas para determinar las acciones tácticas que tienen lugar en las aplicaciones y los sistemas. El **SGRN** se relaciona con otras herramientas de gestión, interactuando a distintos niveles para asistir en la toma de decisiones. [http://en.wikipedia.org/wiki/Business_rule]

Las reglas de negocio son un componente clave en cómo se toman las decisiones en una empresa. En efecto, cada vez que se toma una decisión dentro de una organización es porque se han consultado reglas definidas, las cuales en muchos casos se encuentran escritas en manuales de políticas que los responsables de decisiones deben conocer para desarrollar su gestión cotidiana. Así, las transacciones son completadas por personas basándose en el conocimiento formal o informal de las reglas y políticas de negocio de su organización.

Con un SGRN, los analistas del negocio determinan y escriben la lógica del negocio. Todo lo que necesitan es escribir una regla. Normalmente, una regla de negocio se expresa, con un “Si”, es decir, si existen tales condiciones, “Entonces” deben ejecutarse tales opciones y/o acciones o bien “en caso contrario”, ejecutarse tales otras. La relación entre esas condiciones y las acciones a llevar a cabo crean una regla de negocio, la cual puede o no estar relacionada también con otras reglas. Un SGRN permite que los analistas del negocio puedan ver y entender las reglas sin tener que depender del departamento de informática para realizar las modificaciones.

Las reglas de negocio a menudo están codificadas dentro del software, dificultando su acceso y modificación e incrementando los costes. Gracias a que permite hacer una gestión explícita de las reglas de negocio independientemente del software de las aplicaciones, el SGRN se ha convertido en la tecnología clave para las arquitecturas orientadas a servicios, la gestión de los procesos de negocio y otras iniciativas tecnológicas innovadoras. Con las herramientas

actuales, el usuario dispone de una interfaz amigable para reprogramar por si mismo sus reglas de negocio.

En su relación con la **Inteligencia de Negocio** (*Business Intelligence*), el SGRN tiene dos facetas. Por un lado, cuando el SGRN toma una decisión (como por ejemplo aprobar una solicitud de un producto) añade información que describe cómo o por qué se tomó la decisión. Con ella aporta un elemento crítico para justificar o entender una decisión, yendo más allá del tradicional "aprobado" o "rechazado" y entregando mejores condiciones para el análisis, entender e incluso archivar reglas y referencias para el futuro. De esta forma, las organizaciones pueden superar la brecha entre cómo se cree que deben tomarse las decisiones y cómo son tomadas en la realidad. Otra ventaja es que la inteligencia de negocios basada en un SGRN provee a la organización de un conjunto de reglas de negocio bien definidas y que pueden ser utilizadas en última instancia no sólo para analizar tales decisiones, sino también para reducir el coste y el riesgo asociado a la propia automatización de las decisiones.

Uno de los usos prácticos de esta herramienta lo aplican las entidades bancarias. Por ejemplo, para la concesión de un préstamo, el personal de una oficina utiliza una herramienta apoyada en un **BPM** (Sistema de gestión de procesos de negocio – *Bussiness Process Management*). Este sistema tiene definidos los procesos necesarios y las condiciones que tienen que cumplirse para la concesión del préstamo. Conforme el empleado bancario va avanzando a lo largo del proceso, se van produciendo puntos de bifurcación donde hay que tomar decisiones. En estos puntos, el sistema BPM lanza una consulta al SGRN y este le devuelve la solución para continuar el proceso, en función de los datos disponibles hasta el momento y de las reglas de negocio definidas.

Un sistema de gestión de reglas de negocio incluye como mínimo: un repositorio, permitiendo que las decisiones lógicas sean externalizadas del código fuente de la aplicación. Herramientas, permitiendo tanto a los desarrolladores como a los expertos de negocio a definir y gestionar la lógica de las decisiones. Un entorno de ejecución, permitiendo a las aplicaciones invocar la lógica de decisiones manejadas por el sistema de gestión de reglas y ejecutarlas usando el motor de reglas de negocio. **DROOLS** es una de las tecnologías más ampliamente usadas para representar SGRN. Se detalla en la siguiente sección.

1.2.6 DROOLS

Drools es un sistema de gestión de reglas de negocio con un motor de reglas basado en inferencia con **razonamiento deductivo** (*forward chaining*), más correctamente conocido como sistema de reglas de producción, que usa una mejorada implementación del **algoritmo Rete** (algoritmo de reconocimiento de patrones basado en relacionar cada regla con los hechos de la base de conocimiento activando la regla si procede). Permite expresar de una forma más natural las reglas de negocio interactuando con los objetos de negocio. Provee separación lógica (reglas) y datos (hechos). También provee soporte para la programación declarativa, y es lo suficientemente flexible para expresar la semántica del problema con un lenguaje específico de dominio. [http://www.jboss.org/drools/]

Para especificar las reglas, Drools utiliza el lenguaje de reglas de drools (DRL) usado para especificar las condiciones, acciones y funciones de las mismas, las cuales se puede expresar con distintos lenguajes específicos de dominio (*DSL*), los cuales se asocian a un drl, y también existe la opción de especificar las reglas en una plantilla de cálculo, como Excel. JBoss Rules es un motor de razonamiento que incluye un motor de reglas con encadenamiento hacia delante basado en Drools. JBoss Rules es la versión productiva de Drools.

1.2.7 MAXIMA

Es importante mencionar también esta tecnología que será usada en el proyecto. Máxima es un motor de cálculo simbólico escrito en lenguaje Lisp publicado bajo licencia GNU GPL. [http://es.wikipedia.org/wiki/Maxima]

Cuenta con un amplio conjunto de funciones para hacer manipulación simbólica de polinomios, matrices, funciones racionales, integración, derivación, manejo de gráficos en 2D y 3D, manejo de números de coma flotante muy grandes, expansión en series de potencias y de Fourier, entre otras funcionalidades. Además tiene un depurador a nivel de fuente para el código de Maxima.

Como la mayoría de sistemas algebraicos, Maxima se especializa en operaciones simbólicas. También ofrece capacidades numéricas especiales, como son los números enteros y racionales, los cuales pueden crecer en tamaño sólo limitado por la memoria de la máquina; y números reales en coma flotante, cuya precisión puede ser arbitrariamente larga. Permite el manejo de expresiones simbólicas y numéricas, y además produce resultados con una alta precisión.

2. ORÍGENES Y MOTIVACIÓN

2.1 ORÍGEN DEL PROYECTO

Ahora que ya se tiene una visión más amplia de lo que son los simuladores empresariales y otras herramientas basadas en la inteligencia artificial, así como de cuál es su objetivo y qué ventajas conlleva el uso de estas tecnologías todo se centrará en este proyecto y en el simulador **SIMBA**, ver de dónde se parte y cuál es la problemática que se desea resolver.

El proyecto parte del proceso de simulación de SIMBA. En SIMBA los resultados y la clasificación de las empresas que componen el mercado se obtienen **después de realizar el proceso de simulación**. En este proceso se toma el estado del mercado (el estado es la información, atributos y valores que permiten cuantificar y medir la situación de las empresas unas respecto de otras) en los períodos anteriores y según una serie de cálculos se obtienen las previsiones para el siguiente período. En este proyecto el mercado estará compuesto siempre por **seis empresas** por lo que este número permanecerá constante. Este mercado tiene una serie de atributos llamados "Atributos de Entorno" y otros parámetros como puede ser el IPC, los tipos de interés de préstamos, etc. A su vez cada empresa tiene también una serie de características agrupadas en diferentes secciones (Calidad, Tesorería, etc.) y cada una de estas secciones está compuesta por una serie de atributos que reflejan el estado de la empresa en el período actual y en períodos anteriores.

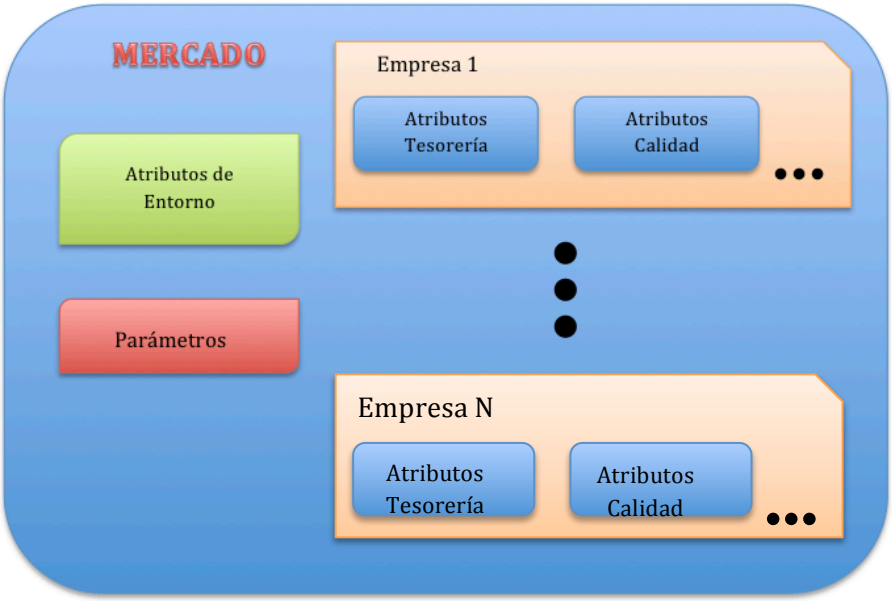


Figura 1: Información del mercado

Toda esta información de la que se dispone es lo que se denomina el *"Histórico"* del mercado con toda la información de los anteriores períodos. La información del Histórico en el periodo actual es la que describe el estado actual del mercado. La información de todo el Histórico está categorizada en lo que se llama la Ontología Genérica que básicamente es un modelo de clases que permite modelar la información.

El propósito del proceso de simulación es ver en qué estado quedará el mercado y las empresas que lo componen en el siguiente periodo de la simulación para poder anteponerse a la situación y tomar unas decisiones u otras sabiendo de antemano cuales son las previsiones.

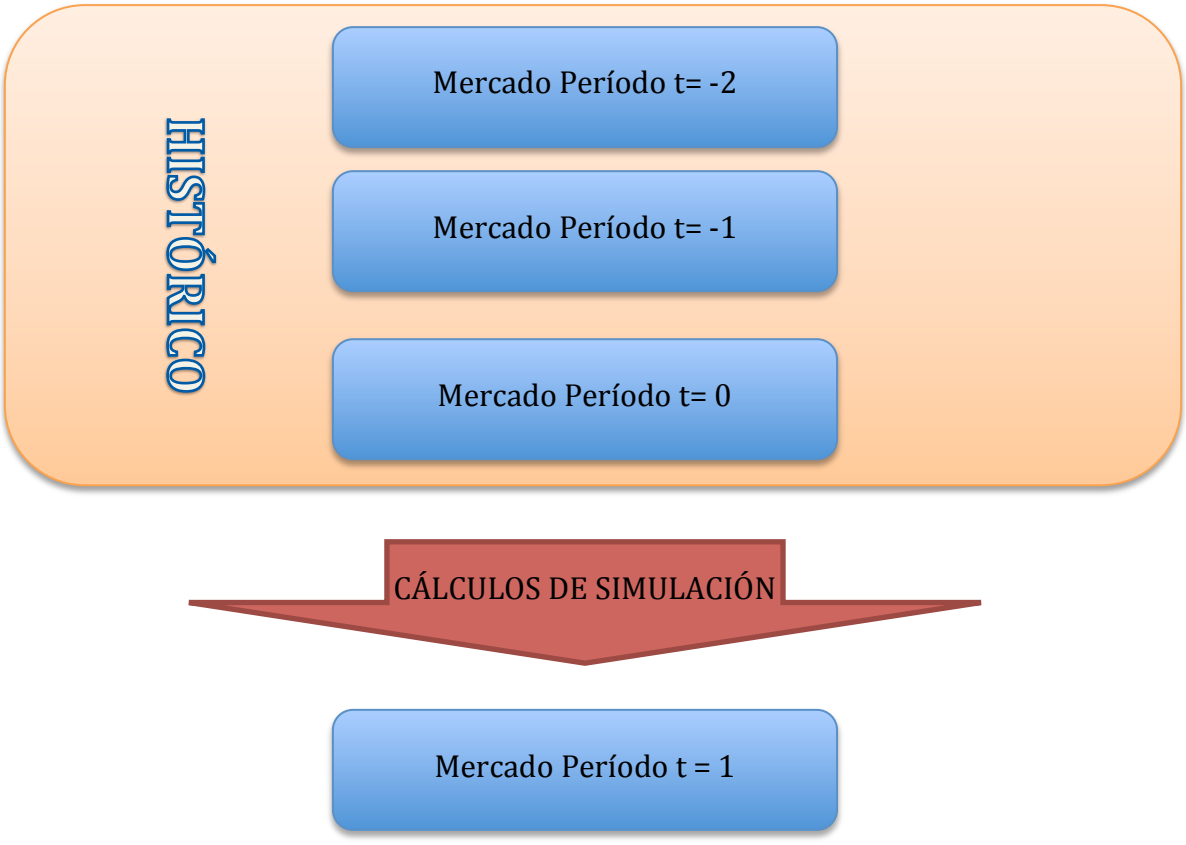


Figura 2: Proceso de simulación

Estas simulaciones van guiadas por una serie de cálculos que son los que determinan el valor futuro de los atributos de las empresas, y es aquí donde este proyecto toma un papel importante. La tecnología que se usa para representar todo este proceso de simulación es CLIPS (mencionado en la sección 1.2.4). CLIPS

como sistema de producción experto está compuesto de una base de conocimiento que incluye hechos y reglas. Mediante los hechos se puede representar la ontología genérica, es decir, el estado del mercado en los períodos previos y en el actual. Y mediante las reglas CLIPS se pueden representar los cálculos que controlan la simulación.

El problema principal es el tiempo que se tarda en modificar una regla existente o crear una nueva al gusto del usuario. Esta herramienta está dirigida principalmente a estudiantes de ciencias empresariales, formadores, directivos, etc., es decir, un perfil de usuario que no está acostumbrado o que quizá nunca haya tratado con un lenguaje de programación. Incluso aunque este tipo de personas dispusieran de un programador / desarrollador para componer los distintas reglas la tarea de modificarlas se antoja bastante ardua y pesada. A la hora de modificar una regla es necesario controlar las variables de restricción, las variables que se usan para controlar el flujo de la ejecución, modificar los atributos que se desean cambiar así como los nuevos operadores que forman el cálculo.

Dependiendo de la complejidad de la regla y de la experiencia del usuario, la modificación de una regla es una labor que puede llevar entre **10 y 30 minutos aproximadamente según estimaciones realizadas**, ya que aparte del cambio de todas las variables anteriores sería necesario probar la regla por separado para ver si da el resultado esperado y luego integrarlo con el resto de reglas.

2.2 OBJETIVOS DEL PROYECTO

Vista la problemática inicial, es aquí donde entra en juego la herramienta desarrollada. El **objetivo global** del proyecto completo desarrollado es crear un sistema que permita **añadir, editar y eliminar las reglas en lenguaje CLIPS y decisiones que componen el sistema de producción usado en el simulador SIMBA**. A su vez estas reglas deben poder ser **categorizadas** en diferentes módulos y dentro de cada módulo debe ser posible **asignar una prioridad** a cada regla.

En este sistema **debe ser posible componer estas reglas de una forma rápida y sencilla** mediante una interfaz parecida a un editor de fórmulas o ecuaciones como el que se ve en la siguiente figura [Fig. 3]. Además, antes de dar por buena una regla ha de comprobarse que todos los elementos que el usuario usa para componer la misma son válidos y la sintaxis de esta es correcta.

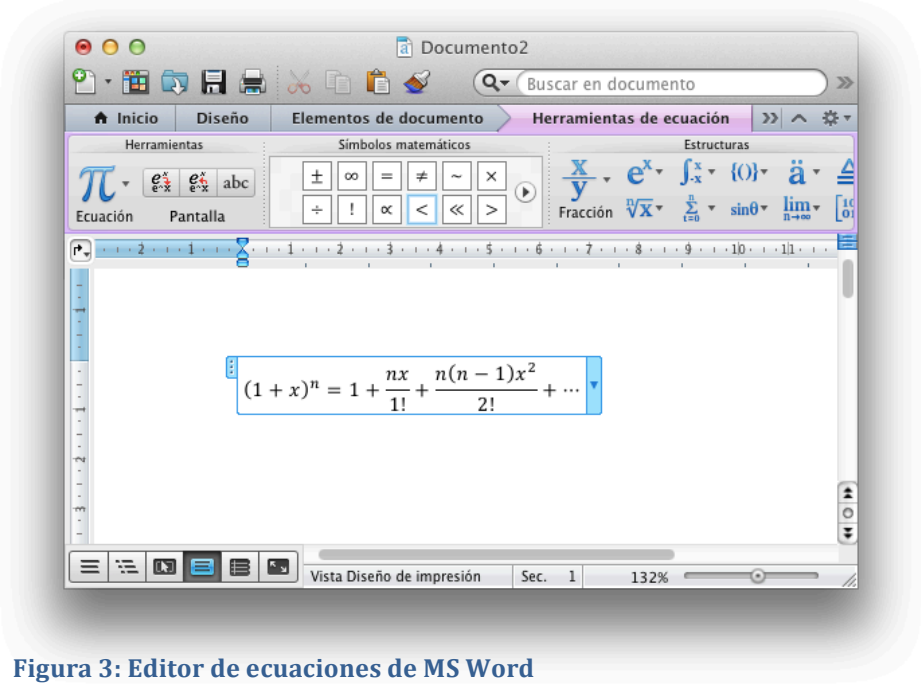


Figura 3: Editor de ecuaciones de MS Word

Finalmente una vez que el usuario ha terminado de trabajar con las reglas del simulador debe poder **obtener dos ficheros con toda la información necesaria** en formato CLIPS. Un fichero contendrá las nuevas reglas que guiarán la simulación y el otro fichero contendrá la nueva ontología genérica. Con ambos

ficheros ya sólo haría falta juntarlos con los otros archivos necesarios para ejecutar la simulación y **obtener los resultados para el nuevo período de simulación**.

Visto el objetivo global del proyecto completo es momento de describir los **objetivos particulares** de la parte del proyecto que se trata en todo este documento.

El objetivo principal de este subproyecto es el de **generar los archivos necesarios a partir de la información creada por el usuario en el editor,** en el formato correcto (CLIPS), para poder **realizar correctamente una simulación** y obtener el estado del mercado en el período futuro. Esto implica generar un archivo con la **categorización de la información** del mercado (ontología genérica) y otro con las **reglas que guiarán la simulación.**

También es cometido de este proyecto el comprobar que cada regla es **léxica y sintácticamente correcta**. Se debe realizar esta comprobación justo después de crearla o editarla y antes de grabarse en el sistema. **En este proyecto se aborda el problema de la comprobación y se proporciona el mensaje de error en caso de que existiese.** Esto se hace así para asegurar que toda regla introducida en la aplicación puede ser traducida.

Otro punto a tener en cuenta en la traducción que es necesario cumplir es que esta hay que realizarla según **el flujo establecido por la prioridad** asignada por el usuario. Esto implica también que las reglas CLIPS deben ser independientes unas de otras para poder editarlas y cambiarlas de orden sin problema alguno.

2.3 HERRAMIENTAS DEL SISTEMA

En esta sección se describen las herramientas software y hardware usadas durante el desarrollo de este trabajo para la implementación, documentación y presentación del proyecto.

HARDWARE

- MacBook Pro 2009
- MacBook Air 2011

SOFTWARE

- Sistema Operativo Mac OSX 10.6 Snow Leopard
- Sistema Operativo Mac OSX 10.7 Lion
- Software de Virtualización Parallels Desktop 6.0 for Mac
- Sistema Operativo virtualizado Ubuntu Linux 11
- Sistema Operativo virtualizado Windows 7
- Microsoft Office for Mac
- Omniplan 2.0.3 (Software de gestión de proyectos)
- Keynote (Software de presentación)
- CLIPS (Linux)
- IDE Netbeans 7.0 for Mac
- MAMP 2.0.1 (Mac, Apache, MySQL, PHP)

3. DESARROLLO DEL PROYECTO

Los inicios de este proyecto se podrían datar a finales del año 2009, aunque no fue hasta bien entrado el 2010 cuando se empezó a contar con suficiente información con la que poder trabajar. Se contaba en un principio con una especificación del tipo de reglas que la herramienta debía ser capaz de generar y un ejemplo compuesto de una serie de ficheros CLIPS con la información necesaria para ejecutar una simulación. Estos ficheros formaban parte del proyecto fin de carrera de otro estudiante (Daniel Sánchez Cisneros) y sirvieron como un primer acercamiento a la herramienta que se debía desarrollar.

3.1 EDITOR DE REGLAS Y TRADUCCIÓN DE REGLAS

Tras recopilar la información antes mencionada se empezaron a dar los primeros pasos estudiando las posibles tecnologías a usar en el desarrollo del proyecto. A la par, se comenzaron a analizar todos los ficheros CLIPS con los que se contaba como ejemplo, qué es lo que hacían, cómo se podían realizar las reglas y los cálculos en un sistema basado en reglas de producción como es CLIPS, el flujo de ejecución de estas reglas, etc. Como más adelante se puede comprobar, el fichero que contiene las reglas del sistema (por defecto denominado generador-predicciones.clp) no tiene nada que ver con el que se partió en un primer momento ya que este se adecuó a los nuevos requisitos del sistema, sin embargo, los resultados de la simulación son exactamente idénticos.

Comprender el modo de funcionar de estos ficheros de partida forma parte de los primeros pasos dados en este proyecto. Y el estudio de diferentes tecnologías que sirviesen para desarrollar el editor de cálculos forma parte de las primeras etapas del proyecto homólogo a este.

A pesar de que cada proyecto se centra en objetivos diferentes la comunicación durante el desarrollo de ambos fue **totalmente indispensable** para el progreso correcto en paralelo ya que las limitaciones y restricciones en una parte podían afectar en gran medida en los avances del otro. Un ejemplo de esto que merece la pena destacar se produjo cuando se comenzó a trabajar con el lenguaje LaTeX (lenguaje usado para la composición de fórmulas matemáticas).

Este iba a ser usado como intermediario en la traducción a las reglas en CLIPS. Pero no fue hasta pasado un tiempo después cuando se comprobó que no se podía trabajar bien con él en el editor debido a problemas en la grabación en la base de datos. Hubo que dar marcha atrás y buscar soluciones alternativas hasta que se optó finalmente por trabajar con el ya mencionado lenguaje Maxima.

En la siguiente sección se muestra de forma superficial la arquitectura del sistema. Se puede obtener así una vista previa de los diferentes componentes que lo integran para poder entender de mejor forma la implicación de cada uno de ellos que se explicará en secciones posteriores.

3.2 ARQUITECTURA DEL SISTEMA

En esta sección se presenta la arquitectura del sistema donde se puede apreciar su estructura y los elementos que lo componen. La entrada de información en el sistema está representada por el usuario que es el que crea y edita los cálculos. La salida estaría representada por los ficheros que se obtienen al finalizar la edición y traducción. Estos dos ficheros son la nueva ontología genérica y las reglas CLIPS que se han creado tras la traducción de los cálculos.

Figura 4: Arquitectura del Sistema

Entre los elementos que componen el **proyecto global** destaca el **editor de cálculos.** Este es la interfaz de la aplicación con la que el usuario interactúa, creando, editando y eliminando cálculos. Cada vez que el usuario acaba de trabajar con un cálculo este se verifica para ver si es traducible. La información se almacena en la base de datos **MySQL**, donde los cálculos se guardan en lenguaje **Maxima**, que ya se mencionó anteriormente.

Toda la aplicación se almacena en un servidor web **Apache Tomcat**. La aplicación permite entre otras cosas la carga de la ontología genérica inicial, la administración de usuarios, la edición de cálculos. Pero de entre todas las funcionalidades soportadas destacan las que se realizan en los paquetes JAVA *OntologiaPackage, GrammarPackage y TranslatorPackage*. Estos contienen el código con la funcionalidad que permite **verificar y traducir los cálculos**. Serán explicados posteriormente.

Por último se devuelve al usuario por medio del editor los archivos con la nueva ontología y las reglas traducidas que podrán ser llevados al entorno de simulación para predecir el nuevo período.

Los elementos con los que se trabaja en este subproyecto son: la **base de datos MySQL** y **los paquetes mencionados contenidos dentro de la aplicación**. La primera es el elemento intermedio entre los dos subproyectos. Es en esta donde se almacenan los cálculos y la ontología. Ha sido modelada atendiendo totalmente a las restricciones impuestas por el proceso de traducción, de forma que se pueda encontrar toda la información necesaria para realizarlo. Los paquetes contiene todo el código Java para cumplir la funcionalidad de **verificación y traducción. El resto de elementos no son ajenos a este proyecto pero quedan fuera del alcance del mismo aunque en varias ocasiones se les deberá hacer referencia.**

3.3 ANÁLISIS DEL TRADUCTOR

A partir de ahora el documento se centrará exclusivamente en la parte de la traducción que es al fin y al cabo la parte que permite crear o modificar las reglas según deseo del usuario. Alguna vez se hará mención a alguna característica del proyecto homólogo ya que hay partes que son comunes. Para comprobar si una regla introducida por el usuario es correcta lo que se hace es "parsear" esa regla y ver si coincide con alguna producción de la gramática. Esta gramática es la que determina el tipo de reglas con las que es posible trabajar por lo que es necesaria para verificar la correcta composición o no de una regla. Si la regla coincide con alguna producción de la gramática se grabará en la base de datos MySQL, en caso contrario, se advertirá al usuario con un mensaje de error apropiado indicando el motivo y se permanecerá a la espera de que lo corrija. El primer paso en el análisis es comprender cómo se realiza una simulación.

3.3.1 PRIMERA SIMULACIÓN

El primer acercamiento a la traducción fue con los archivos CLIPS que se nos proporcionaron como ejemplo de cómo se realiza una simulación. Fue en estas primeras simulaciones donde se empezó a tener conocimiento de cómo funciona la ejecución de las reglas. Para realizar una correcta simulación son necesarios los siguientes archivos:

- **Entrada.in:** Este archivo sólo define el número de empresas que componen el mercado y el período actual de la última simulación realizada. Para cada empresa se detalla el **nombre** de la misma y su **identificador** (ID). Este archivo no se modificará, ni se creará más de una vez. Sólo debe estar en su ruta correspondiente para cuando se realiza la simulación.

- **Historial.clp:** Este archivo contiene toda la información histórica de anteriores simulaciones. En él se encuentran los atributos de cada empresa categorizados por diferentes módulos. Aparte de la información para cada empresa también existen otra serie de atributos como los atributo de ***Entorno*** que también van variando en el tiempo. Otros atributos que existen en el archivo de Histórico son los que corresponden a las clases ***Pesos***, ***Decisiones*** y ***Prefijos***. Este archivo tampoco se va a crear y/o modificar.

- **Predicciones.out:** Este archivo muestra todos los resultados de la simulación ordenados por módulo. Sólo se usará para comprobar que los resultados obtenidos son los esperados.

- **Ontologia-genérica.clp:** Este archivo sí tiene importancia para la herramienta a desarrollar. La Ontología define las clases y los atributos (*slots*) que aparecen en el Histórico (Historial.clp). Es el único medio que existe para conocer que tipo de información se guarda en el Histórico y por lo tanto **es imprescindible para que la simulación no falle que la información que aparece en el Histórico case totalmente con la que aparece en la Ontología**. La ontología informa de los atributos con los que el usuario puede trabajar y usar para componer cálculos y reglas por eso aparece en el panel derecho del editor de cálculos de la aplicación.

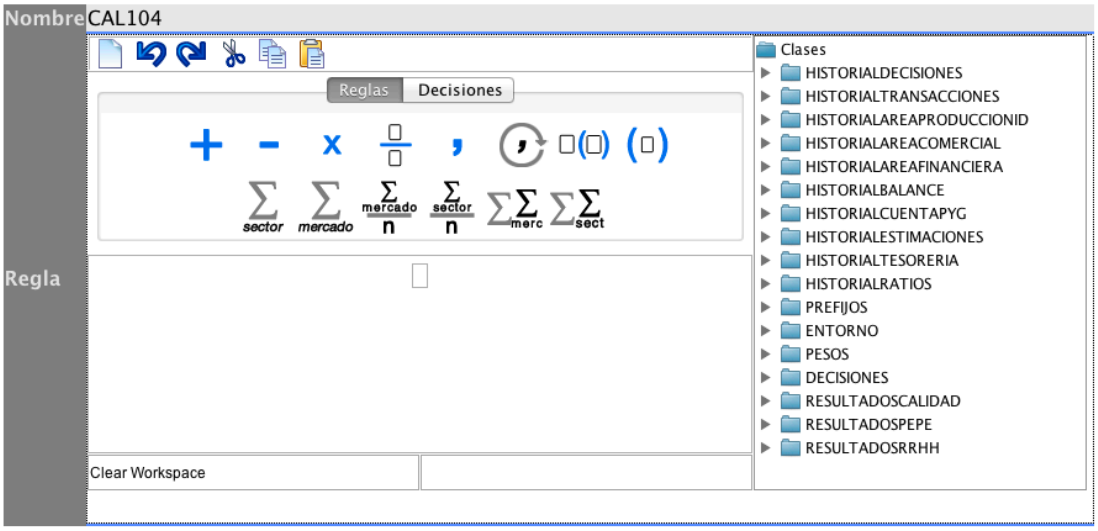


Figura 5: Editor de cálculos

- **Generador-predicciones.clp :** Este archivo es el de más importancia para la herramienta, junto con el de la Ontología. **Define todas las reglas en formato CLIPS que darán lugar a la creación de los cálculos de la simulación.** Este fichero está estructurado en diversos módulos y cada uno de ellos tiene los diferentes cálculos en su interior. A continuación se explicará más en detalle.

El generador-predicciones.clp contiene todas las reglas CLIPS necesarias para **controlar el flujo de ejecución de la simulación** y para formar los cálculos de la misma. Además de las reglas que crean los cálculos también están las reglas iniciales que permiten, por ejemplo, definir el "rango" de una empresa (sector alto, medio o bajo) dependiendo de un atributo discriminante, o las reglas que tratan con el fichero donde se guardarán los resultados. En el archivo original de donde se partió la simulación estaba muy cerrada, no estaba pensada para modificarse por lo que la reutilización de valores creados en una regla inicial preparatoria era algo muy común. De igual forma, el flujo de ejecución en su simulación hacía que ciertas reglas **no fuesen independientes unas con otras** lo que está bien si se sabe con certeza que nada va a cambiar pero algo totalmente inútil para la solución que se buscaba en este proyecto. Este esquema resume muy bien el flujo de simulación con el que se contaba en un principio :

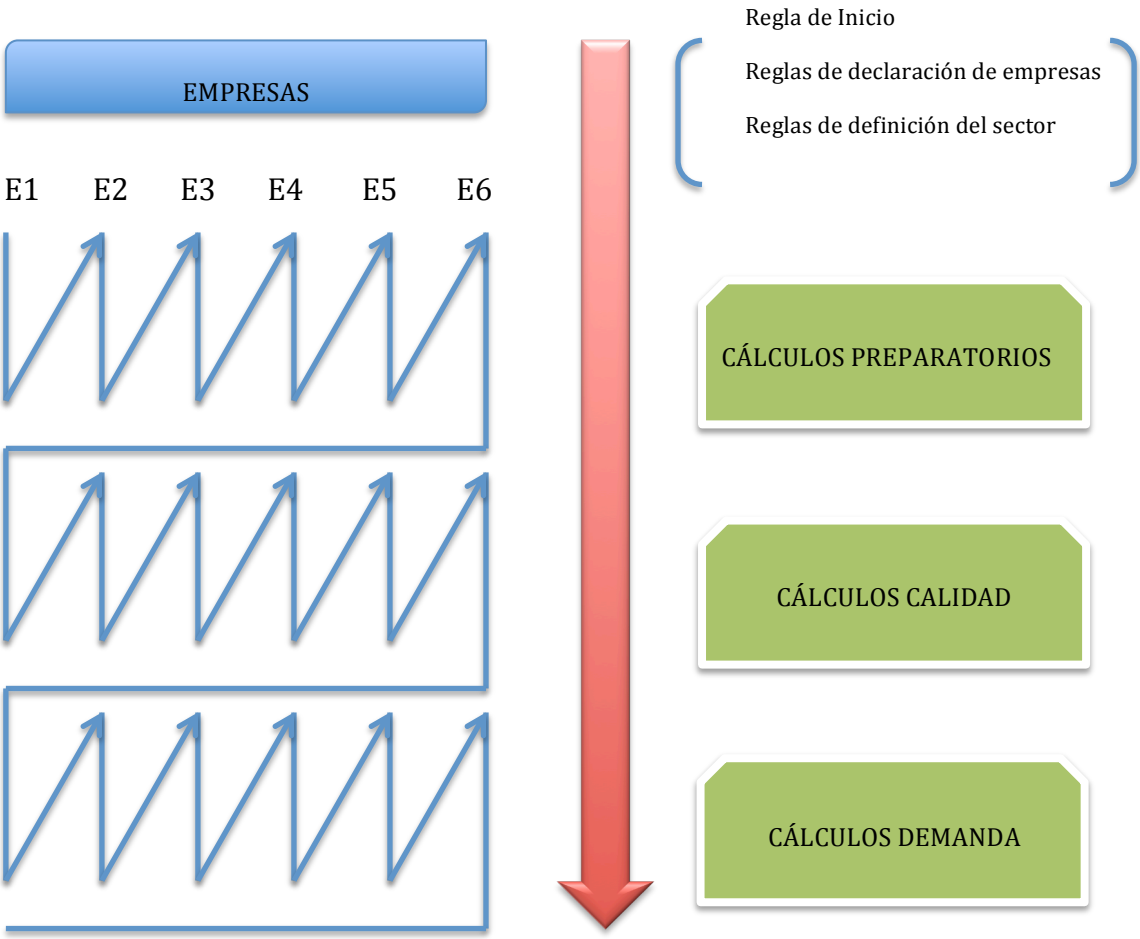


Figura 6: Flujo de ejecución inicial

Como se puede apreciar se tienen las seis empresas que componen el mercado y a la derecha los diferentes módulos que sirven para clasificar y agrupar las reglas. En el flujo inicial se tenían una serie de **reglas iniciales** usadas para iniciar el manejo del fichero en el que se guardarán los resultados, para declarar las empresas que van a participar en la simulación y establecer el sector al que pertenecen según un atributo discriminante (sector alto, medio o bajo). A continuación se ejecutan una serie de **reglas preparatorias** (o cálculos cero) que se dedican a realizar cálculos que luego se aprovecharán para reglas posteriores. En las simulaciones con las que se trabajaba abundaban los cálculos de sumatorios de atributos donde son necesarias bastantes reglas para obtener el valor final del cálculo (algunos de estos sumatorios necesitan más de 1000 líneas de código CLIPS), por eso es totalmente comprensible que se quisiera optimizar el proceso creando una serie de reglas preparatorias donde se obtienen algunos valores que pueden reutilizarse posteriormente. Sin embargo esta solución no parecía viable y se optó porque cada **cálculo esté compuesto de las reglas CLIPS que sean necesarias para hacer que el cálculo sea totalmente independiente de otros cálculos anteriores.**

Otra diferencia sustancial comparada con la solución que finalmente se aplicó está en la **dirección del flujo de ejecución**. Como se puede ver en el esquema anterior, cuando se entra a ejecutar las reglas de un determinado módulo estas reglas se ejecutarán para cada empresa. Primero se toma la empresa uno y se ejecutan **todas las reglas del módulo**, después se toma la empresa dos y se vuelven a ejecutar todas las reglas de ese módulo, así hasta que **se pasa por todas las empresas**. Una vez que se ha pasado por todas las empresas se ejecutan las reglas que "imprimen" los resultados de ese módulo en el fichero ***resultados.out***. Posteriormente se pasará al siguiente módulo donde se seguirá con el mismo procedimiento hasta el final. Se obtienen así los resultados para cada empresa, que es al fin y al cabo lo que se busca, pero …, **¿qué ocurre si en un módulo existe un cálculo que necesita de otro anterior del mismo módulo?** Pues que como todavía no se tienen todos los resultados para todas las empresas **no es posible crear un cálculo que necesite de uno anterior del mismo módulo**. Este problema se resolverá con un rediseño del flujo de ejecución. Aunque esto se estudiará en la sección de Diseño.

3.3.2 CÁLCULOS Y DECISIONES

En esta sección se entra más en profundidad a descubrir los **tipos de cálculos** que se podrán crear, eliminar y modificar en el editor. La variedad de cálculos que se pueden componer y traducir desde el editor obedecen a las especificaciones que se proporcionaron en un primer momento. En la herramienta debería ser posible crear y editar los cálculos que se hacían en el **fichero de simulación de prueba original**. Una vez acabada la implementación se puede afirmar que la variedad de cálculos que se pueden hacer **es mayor** a la que se propuso inicialmente, permitiendo así mucha más flexibilidad al usuario indicándole además cuando ha cometido un error. Los cálculos que se pueden formar podrían clasificarse dentro de estos cuatro tipos:

- **Cálculos aritméticos simples**
- **Sumatorios**
- **Decisiones**
- **Redondeos**

Los cálculos aritméticos simples son los típicos cálculos compuestos por **operandos y operadores.** Los operadores son la suma, resta, multiplicación y división. Los operandos son cualquier atributo que aparezca en la ontología (también puede ser el resultado de un cálculo anterior) o cualquier número, pudiendo este ser negativo o positivo, con cifras decimales o sin ellas. **El elemento separador de las cifras decimales es el punto** no la coma. Cualquier error que el usuario introduzca será advertido mediante un mensaje de aviso pudiendo este indicar un **error de tipo léxico** (carácter que no existe) o **sintáctico** (por ejemplo, dos operadores seguidos).

A primera vista puede parecer extraño que los cálculos sumatorios no sean clasificados dentro del grupo de operaciones aritméticas, pero es que estos sumatorios tienen una característica peculiar. Los sumatorios corrientes están compuestos de un índice con un valor inicial y un valor final, este índice es el que mide el número de iteraciones que se van a realizar, es decir, el número de sumas del valor que acompaña al sumatorio. Pero en los sumatorios que aparecen en este proyecto existe más información añadida. En todos los sumatorios junto al símbolo Σ aparece la palabra sector o mercado, **esto indica si se debe sumar el atributo de todas las empresas, o sólo de las empresas que son del mismo**

sector que la empresa de la que se está haciendo el sumatorio. El sector o "rango" viene determinado por el valor de un atributo que pertenece a la clase de la ontología ***HISTORIAL AREA COMERCIAL***. Dependiendo de este valor la empresa será considerada como de sector alto, medio o bajo. Las reglas que asignan un rango a cada empresa son lanzadas al principio de la simulación. Como se puede ver, el sumatorio, aparte del contenido "aritmético" lleva interiormente una restricción asociada, razón por la cual se tratan de forma diferente al resto de cálculos aritméticos. Estos son los seis tipos de sumatorios con su significado:

Sumatorio de sector de un atributo. Este sumatorio calculado para una empresa tomará el atributo que venga indicado y los sumará con los atributos del mismo nombre de las empresas que estén en su mismo sector.

Sumatorio de mercado de un atributo. Es exactamente igual que el anterior solo que sumará el atributo que se indique de todas las empresas, sean del mismo sector o no.

Sumatorio de los cuatros anteriores periodos del sumatorio de sector de un atributo. Es igual que el primero con la diferencia de que se calcula con los cuatro periodos anteriores ((t), (t-1), (t-2), (t-3)), es decir, se hace un cálculo como el primero para los cuatro periodos anteriores y al final se realiza la suma de todos.

Exactamente igual que el anterior pero en vez de realizar el sumatorio de sumatorios del atributo cuyas empresas están en el mismo sector, se hará para todas las empresas por igual, es decir, se calcula el sumatorio visto en segundo lugar para los cuatro periodos anteriores y luego se suma todo.

Este sumatorio es igual que el primero pero se dividirá entre el número de empresas que hay en el mismo sector de la empresa para la que se está haciendo el cálculo.

En este se realiza la suma del valor del atributo que se indique para todas las empresas y se divide entre el número total de empresas del mercado.

Las decisiones son un tipo de cálculo que permite al usuario asignar un valor determinado dependiendo de si se cumple una condición o no. Este nuevo valor debe ser un número entero (decimal o no) y la comparación de la condición debe ser entre atributos de la ontología genérica. Los comparadores aplicables son “>=”, “<=” ó “=”. Si el usuario comete cualquier error a la hora de escribir la decisión recibirá un mensaje de error en el que se mostrará donde está el problema.

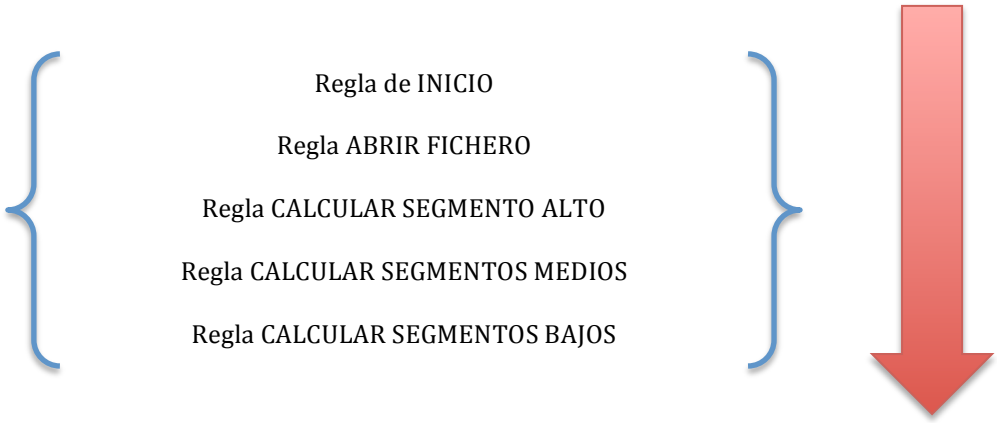
Por último, **el redondeo**. El redondeo es un tipo de cálculo que se dio en la especificación de los cálculos que debían poder traducirse. Su función es la de eliminar las cifras decimales del valor del atributo que va entre los paréntesis del cálculo aplicando un redondeo. Está representado por este símbolo en el editor:

Posteriormente se verán las gramáticas que se han usado para la verificación y traducción de reglas. La gramática **permite definir que producciones (cálculos) son correctas y cuales no**, y si la estructura del cálculo coincide con una producción de la gramática, poder realizar los procedimientos necesarios para transformarlo en las reglas CLIPS que sean necesarias. Para realizar la verificación de un cálculo y la traducción se ha usado un analizador léxico y un analizador sintáctico. El analizador léxico permite reconocer los *token* y ver cuales no son correctos (y emitir errores de léxico), el analizador sintáctico permite comprobar la estructura del cálculo.

3.4 DISEÑO DEL TRADUCTOR

En esta parte de la documentación se describe el proceso de diseño de la solución, aunque podría llamarse también proceso de rediseño ya que se centrará en **como redefinir el flujo de ejecución de las reglas** para que se adapte a la solución a la que se desea llegar. Como se vio de manera superficial en la sección de Análisis, el flujo de ejecución estaba profundamente optimizado para la simulación cerrada que se quería conseguir en un primer momento. Esto no es malo, si es eso lo que se quiere, pero para el objetivo de este proyecto no era una opción válida ya que **es necesaria una independencia total de unas reglas con otras.** El otro problema era el de poder usar en un cálculo el resultado de otro anterior del mismo o de otro módulo. A continuación se presenta el nuevo flujo diseñado.

Todo comienza con la **regla INICIO**. Esta regla se encarga de dar un nombre al fichero donde se guardarán los resultados. También es la responsable de crear las instancias de las clases RESULTADOS que **son las que guardarán los slots que contienen los valores resultantes de los cálculos.** Después está la regla **abrirfichero** que se encarga de abrir el fichero donde se guardarán los resultados (Predicciones.out) en modo escritura. Por último, en estas reglas iniciales están las que se dedican a asignar un sector a cada empresa (Alto, Medio o Bajo). El atributo discriminante usado para ello es el **RCALI que pertenece a la clase HISTORIAL AREA COMERCIAL.** Si el valor de RCALI para una determinada empresa es inferior a 4.0 esta pertenecerá al sector bajo, si está entre 4.0 y 7.0 será una empresa de categoría media. Si el valor es superior a 7.0 será una empresa del sector alto.



Las reglas que continuarán ejecutándose difieren mucho de las que se partieron en un principio. En el siguiente diagrama se puede observar gráficamente el nuevo flujo de ejecución.

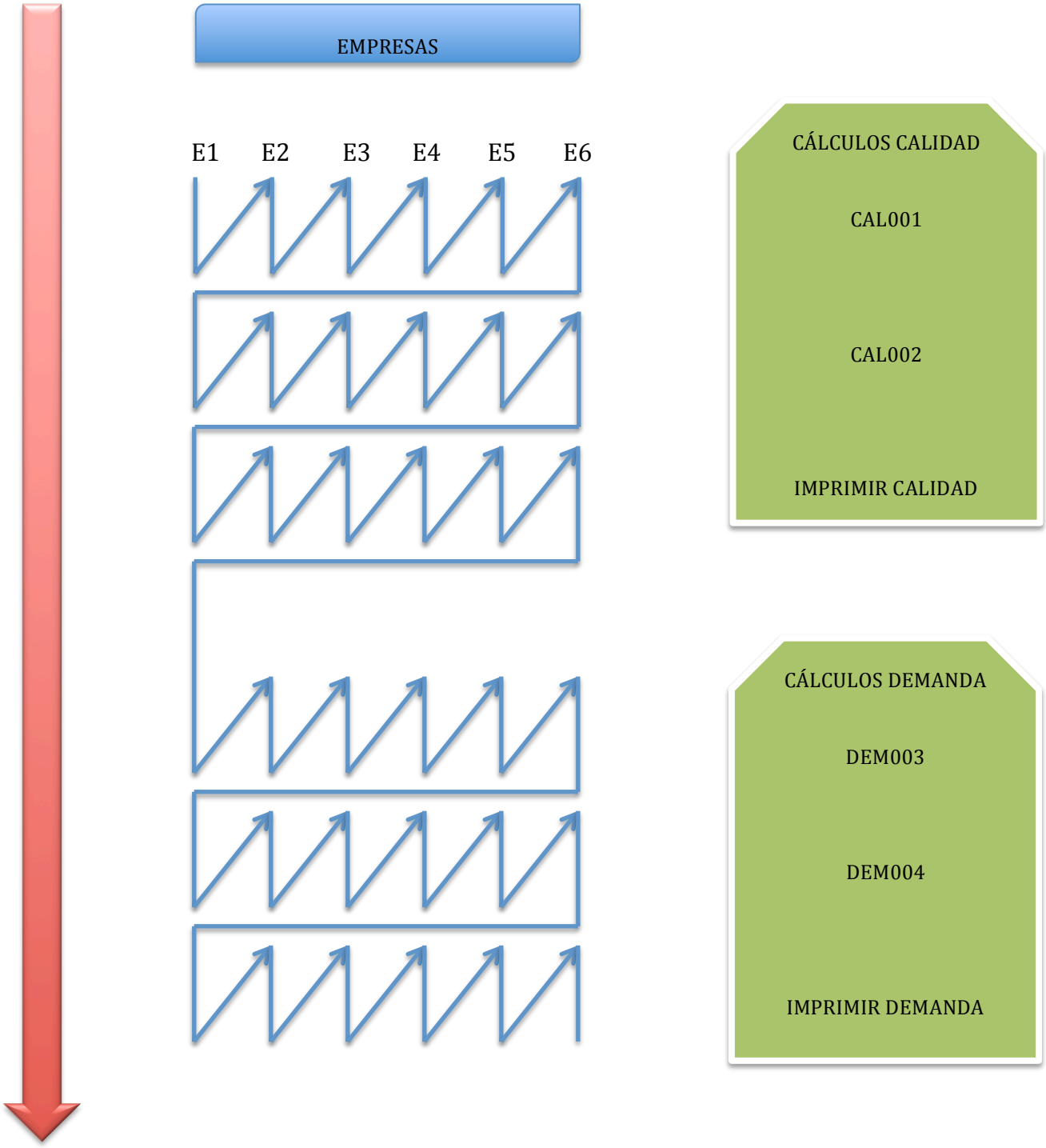


Figura 7: Flujo de ejecución modificado

Como se puede ver, ahora igual que antes se comienza por la primera empresa en el primer módulo, pero en vez de recorrer todas las reglas para esa empresa lo que se hace es tomar el primer cálculo y recorrer todas las empresas. De esta forma se estará obteniendo todos los resultados del primer cálculo para todas las empresas, así que si el segundo cálculo requiere conocer el resultado del primero ya estarán todos los datos a su disposición.

Implementar este flujo de ejecución requiere de métodos de control de iteraciones y adición de otras reglas CLIPS suplementarias que no serán detalladas en profundidad. Básicamente se usa **una variable que controla por qué cálculo dentro de un módulo se esta pasando** y otra variable (una por cada módulo) que **lleva una cuenta del número de empresas para las que se ha ejecutado ese cálculo.** Cuando el número de empresas llega a seis (que es el número de empresas del mercado) se ejecuta una regla (**ReiniciarEquipos**) que inicializa el número de empresas a uno y se pasa al siguiente cálculo.

Este proceso se repite para todos los cálculos (sean del tipo que sean: aritméticos, decisiones, sumatorios). Una vez que se ha pasado por todos los cálculos del módulo se procede con las reglas "*imprimir*" (IMPRIMIR_NOMBREMÓDULO). Estas reglas tienen como misión el tomar todos los *slots* de la clase creada con los resultados para ese módulo y escribirlos en el fichero "Predicciones.out" que es el que contiene los resultados de los cálculos.

Todo este ciclo continuará hasta que ya no existen módulos, o lo que es lo mismo que ya no haya reglas CLIPS que ejecutar.

El proceso de traducción de los cálculos en el editor (que están guardados en lenguaje Maxima dentro de la base de datos) **se inicia cuando el usuario ha acabado de crear y editar todos los cálculos** y va a la pestaña "CLIPS" dentro de la aplicación. . Ahí tiene la opción de acceder a los ficheros con las reglas y la ontología si fueron creados anteriormente, sino, puede optar por crearlos de nuevo. Si es la primera vez que se accede al editor sólo aparecerá la opción de *traducir*. Cuando se selecciona esta acción es cuando se desencadena todo el proceso implementado en este proyecto.

Mencionar también el diseño de la BBDD. **Esta BBDD es la vía de comunicación de información entre el editor y el traductor** por eso es importante reseñarla y mostrar un esquema de su estructura. Ha sido diseñada para almacenar toda la información necesaria para poder realizar la traducción. Es de aquí de donde se carga la Ontología con la que se va a trabajar, que es necesaria para ir obteniendo la información de los atributos y clases. Se cargan también las reglas y se clasifican por módulo para empezar traduciendo el primer cálculo del primer módulo.

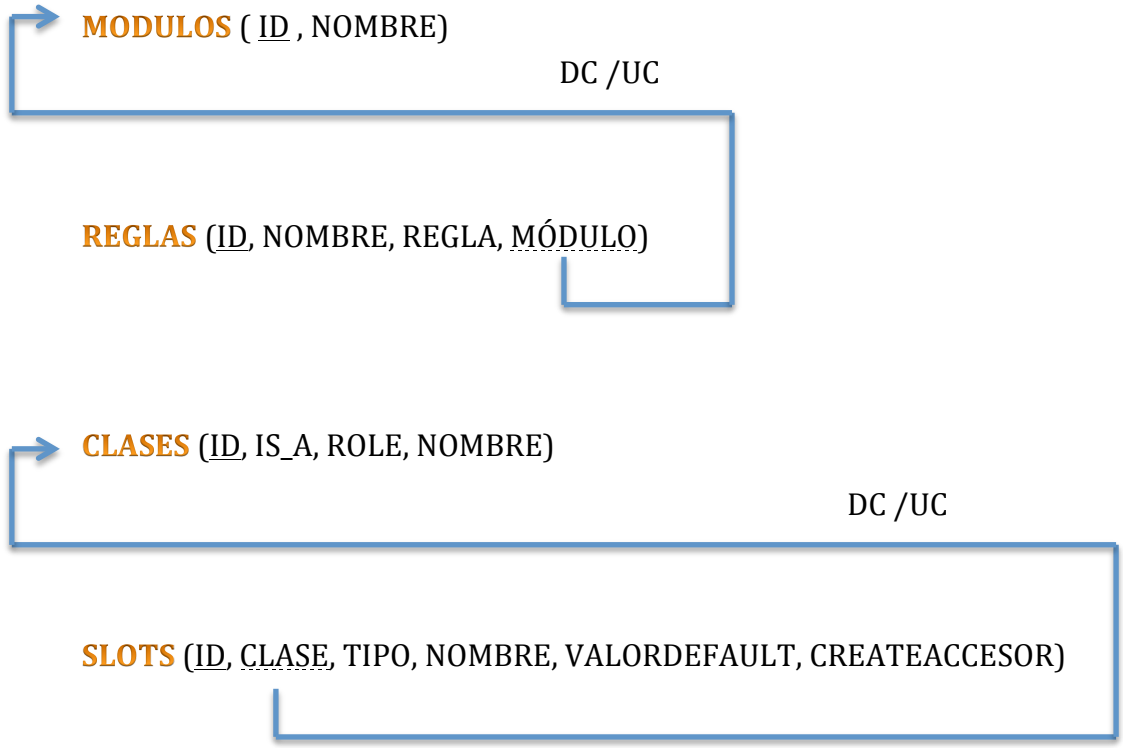


Figura 8: Diseño de la base de datos

Cada módulo tiene su **identificador** (ID) y su **nombre**. El ID es único por eso es la PRIMARY KEY.

Cada regla tiene su **identificador** (ID), un **nombre** (que serán las tres primeras letras del módulo seguidas del ID), **la regla** en sí misma (escrita en formato Maxima) y el **ID del módulo** al que pertenece. El ID de la regla es único por eso es su PRIMARY KEY. Si se borra un módulo se borran todas las reglas de ese módulo. Si se actualiza la información del módulo los cambios se verán reflejados en la tabla de reglas.

Cada Clase de la Ontología tiene un **identificador** (ID), un **campo IS_A** (para indicar de que clase derivan estos objetos Clase, en la mayoría será INITIAL-OBJECT), **ROLE** (para definir si se pueden generar instancias de ellas o

no, en este proyecto todas serán "concrete") y el **nombre de la clase.** Debido a que el campo ID es único este será la PRIMARY KEY.

Cada Slot tiene un **identificador** (ID), pertenece a una clase, es un tipo de dato (Integer, String,…), tiene un **nombre** que lo identifica, tiene un **valor por defecto** y una **propiedad** "CreateAccessor" que sirve para especificar el tipo de acceso que se permite. El ID es único por eso es su PRIMARY KEY. Si se borra una clase se borran todos los Slots que pertenecen a ella. Si se borra la información de una clase se actualiza también el campo Clase de la tabla de Slots.

3.4.1 MODELO DE CLASES

En esta sección se presenta el diseño realizado con las distintas clases que componen los tres paquetes que conforman la solución final. En la parte de implementación de esta memoria se estudiará cual es la función de cada clase. Debido a que la extensión del modelo de clases es demasiado grande esta se presenta en la sección 8.4 de los apéndices. Se adjunta también en la sección 8.5 la documentación JavaDoc generada por el entorno de desarrollo donde se describe con más detalle cada método y atributo de las clases.

3.5 IMPLEMENTACIÓN

En esta sección se presenta toda la documentación relativa a la implementación que ha sido necesaria para desarrollar el sistema comenzando con las tecnologías que se han utilizado.

3.5.1 TECNOLOGÍAS DE DESARROLLO

La implementación de este proyecto ha sido llevada a cabo con las siguientes tecnologías:

- **Netbeans (Versión 6.8).** Netbeans es un entorno de desarrollo integrado (IDE) hecho principalmente para el lenguaje de programación JAVA, es un producto libre y gratuito sin restricciones de uso. Se ha usado para crear las clases que van dentro del traductor, tanto como para comprobar los cálculos como para traducirlos.

- **JLex.** Es un analizador léxico parecido a **LEX**, el cual toma una cadena de caracteres como entrada y lo convierte en una secuencia de *tokens*.

- **CUP.** Es un generador de **analizadores sintácticos LALR** en Java el cual recibe de entrada un archivo con la estructura de la gramática y su salida es un *parser* escrito en Java listo para usarse.

- **MAMP.** Es el conjunto de programas software comúnmente usado para desarrollar sitios web dinámicos sobre sistemas operativos **Apple**. Integra el servidor web **Apache**, el sistema gestor de bases de datos **MySQL** y **PHP**. Se ha usado para alojar la base de datos desde la que se carga la ontología y los cálculos.

- **Tomcat.** Funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta. **Tomcat** implementa las especificaciones de los servlets y de **JavaServer Pages** (JSP).

Las dos últimas herramientas son usadas por el editor pero forman parte también de la implementación del traductor pues han sido necesarias para almacenar los cálculos y la ontología además de para comprobar que la integración entre las dos partes ha sido correcta.

Antes de empezar, es necesario aclarar que todo lo que se comenta aquí es una breve descripción de qué es lo que se realiza en la implementación. No se pretende entrar en demasiado detalle ya que no es relevante para entender todo lo que se ha realizado en este proyecto. Se describirá superficialmente el proceso de **verificación y traducción** y se detallarán las clases que intervienen y su función. Si se necesita o se quiere entrar en más detalle de la implementación, en los apéndices se encuentra la **documentación JavaDoc** generada por el propio **Netbeans** donde se pueden ver todas las clases y los métodos que participan. Si esto no fuese suficiente siempre se puede recurrir al código fuente en busca de los detalles más recónditos.

Las clases Java encargadas de verificar que un cálculo introducido en el editor es correcto están contenidas dentro del paquete "***GrammarPackage***". Dentro del paquete "***TranslatorPackage***" están las clases que traducen el cálculo en formato **Maxima** a las reglas en CLIPS. Ambos paquetes hacen uso de "***OntologiaPackage***". Este paquete contiene las clases necesarias para representar la ontología genérica la cual está siempre en memoria para recurrir a ella en cuanto sea necesario. Este paquete incluye las siguientes clases:

- **DBConnector:** Clase encargada de conectar con la base de datos MySQL para cargar toda la información.
- **OntologiaClass**: Clase encargada de almacenar la información de toda la ontología.
- **ModuloClass**: Clase encargada de representar la información de un módulo.
- **ReglaClass**: Clase encargada de representar la información de una regla.
- **ClaseClass**: Clase encargada de representar la información de una clase de la ontología.
- **SlotClass**: Clase encargada de representar la información de un slot (atributo de la ontología).

3.5.2 DESARROLLO DEL VERIFICADOR DE CÁLCULOS

Las clases que contienen los métodos encargados de comprobar si un cálculo es correcto están dentro del paquete "***GrammarPackage***". Este paquete está compuesto de las siguientes clases:

- **ComprobarRegla.class**: Este paquete contiene el método comprobarRegla(regla). Este método recibe el cálculo del editor, realiza una serie de comprobaciones preliminares y posteriormente llama al analizador léxico y sintáctico.
- **Comprobaciones.class:** Esta clase contiene los métodos que sirven para comprobar si los sumatorios llevan un atributo correcto, si el atributo pertenece a una clase HISTORIAL (es decir, si existen atributos con el mismo nombre pero para distintos períodos), etc.
- **Lexer.class:** Esta clase contiene el analizador léxico creado por JFlex.
- **Sym.class:** Esta clase contiene una definición de los tipos de *token* que puede encontrar el analizador léxico.
- **Parser.class:** Esta clase contiene el analizador sintáctico creado por JCup.
- **LexError.class:** Esta clase lanza un error de tipo léxico indicando en el mensaje la columna donde se encuentra el error.
- **SyntaxError.class:** Esta clase lanza un error de tipo sintáctico indicando en el mensaje la columna donde se encuentra el error.
- **AttributeException.class:** Clase usada para lanzar una excepción propia en la que indicamos el mensaje de error que se debe mostrar. Se usa para lanzar excepciones cuando no se encuentra un atributo de la ontología o errores parecidos.

En el editor, en el momento en el que se ha acabado de crear o editar el cálculo se selecciona la opción de grabar en la base de datos. Justo antes de grabar se llama al método **comprobarRegla(regla)** que está dentro de **ComprobarRegla.class.** Si el cálculo es correcto se grabará en la base de datos sin problema, si es incorrecto el método habrá lanzado una excepción que será capturada en el editor. Esta excepción tiene un mensaje de error que indica al usuario qué error ha cometido y donde. Los mensajes de error se sucederán hasta que la regla introducida sea correcta.

En el método **comprobarRegla(regla)** se realizan una serie de comprobaciones antes de analizar léxicamente el cálculo, cómo ver si se ha optado por grabar en la base de datos sin haber escrito nada (cálculo vacío). Finalmente se llama al parser, (**Parser.class**) el cual necesita del analizador léxico (**Lexer.class**). El analizador léxico identifica y descompone los *token* que recibe por la entrada en formato MAXIMA y lo hace atendiendo a la especificación definida a continuación.

ANÁLISIS LÉXICO

Declaraciones de expresiones regulares para identificar ciertos *token*:

```
LineTerminator = \r|\n|\r\n          // Identifica fin de línea
WhiteSpace     = {LineTerminator} | [ \t\f]  //Identifica espacio en blanco
sumatorio1 = sum\(.+,sector,0,0\)  //Expresión regular sumatorio tipo 1
sumatorio2 = sum\(.+,mercado,0,0\) //Expresión regular sumatorio tipo 2
sumatorio3 = sum\(.+,sector,0,0\)\/n //Expresión reg. sumatorio tipo 3
sumatorio4 = sum\(.+,mercado,0,0\)\/n //Expr. reg. sumatorio tipo 4
sumatorio5 = sum\(sum\(.+,sector,0,0\),t,-3,t\) //ER  sumatorio tipo 5
sumatorio6 = sum\(sum\(.+,mercado,0,0\),t,-3,t\) //ER sumatorio tipo 6
atributo = [A-Za-z]+_?[0-9]?[A-Z]?\(?t-?[0-9]*\)? // ER identifica atributos
calculo = [A-Za-z]{3}[0-9]{3}\(?t-?[0-9]*\)? // ER identifica cálculos
numero = [0-9]+(\.[0-9]+)? //Expresión regular identifica números
```

Acciones llevadas a cabo cuando se encuentran los token

//Cuando se encuentra un sumatorio se comprueba si su atributo existe y se devuelve un *token* de tipo sumatorio

```
{sumatorio6}     {comprobarSumatorio6(Ontologia, yytext()); return symbol(sym.SUM6, yytext());}
{sumatorio5}     {comprobarSumatorio5(Ontologia, yytext()); return symbol(sym.SUM5, yytext());}
{sumatorio4}     {comprobarSumatorio4(Ontologia, yytext()); return symbol(sym.SUM4, yytext());}
```

```
{sumatorio3}    {comprobarSumatorio3(Ontologia, yytext()); return symbol(sym.SUM3, yytext());}
{sumatorio2}    {comprobarSumatorio2(Ontologia, yytext()); return symbol(sym.SUM2, yytext());}
{sumatorio1}    {comprobarSumatorio1(Ontologia, yytext()); return symbol(sym.SUM1, yytext());}
"REDONDEAR"     { return symbol(sym.REDONDEAR);}

// Cuando se encuentra un atributo o un cálculo anterior se comprueba si existe en la ontología y se devuelve un token indicando que es un atributo o cálculo
{atributo}      { comprobarAtributo(Ontologia, yytext());  return symbol(sym.ATR, yytext());}
{calculo}       {comprobarCalculo(Ontologia, yytext());   return symbol(sym.CAL, yytext());}

// Cuando se encuentra un símbolo se devuelve el token que lo identifica
";"             { return symbol(sym.SEMI);  }
"+"             { return symbol(sym.PLUS);  }
"-"             { return symbol(sym.MINUS);  }
"*"             { return symbol(sym.TIMES);  }
"/"             { return symbol(sym.DIVIDE); }
"<"             { return symbol(sym.MENOR);  }
"="             { return symbol(sym.IGUAL);  }
">"             { return symbol(sym.MAYOR);  }
"("             { return symbol(sym.LPAREN); }
")"             { return symbol(sym.RPAREN); }
"if"            { return symbol(sym.IF);    }
"then"          { return symbol(sym.THEN);  }
{numero}        {return symbol(sym.NUMBER, new Float(yytext())); }
```

Aquí se han podido ver las acciones que se realizan cuando se localizan los *token* que se pueden encontrar en un cálculo. Por ejemplo, si se encuentra un sumatorio se comprobará que el atributo que va dentro pertenece a una clase con información para todas las empresas (sino no se podría hacer el sumatorio de empresas). Se devuelven los identificadores para cada *token* (los mismos que

aparecen en Sym.class). Si un *token* no aparece clasificado es cuando se lanza un mensaje de error indicando que existe un error léxico en el cálculo introducido.

ANÁLISIS SINTÁCTICO

Después del análisis léxico es cuando se realiza el análisis sintáctico. El parser sintáctico es creado mediante JCup. Al igual que JFlex este parser se crea especificando la gramática que debe analizar. La gramática se detalla a continuación y es la que define todos los posibles cálculos que el sistema puede crear, editar y traducir. Cuando se encuentra un cálculo que no coincide con una producción de la gramática es cuando se lanza una excepción indicando un error de tipo sintáctico, indicando además en el editor en qué parte del cálculo se ha producido este error. La gramática es la siguiente:

```
// Una producción de la gramática puede ser una serie de expresiones, una expresión, una decisión o un redondeo

  expr_list ::=   expr_list expr_part

            | expr_part

            | decision

            | redondeo


// Un Slot puede ser un atributo de la ontología o un cálculo realizado anteriormente

  slot ::= CAL | ATR


// Un operador de comparación puede ser el "=", el ">=" ó el "<="

  comparador ::= IGUAL | MAYOR IGUAL | MENOR IGUAL


// Un número puede ser un número positivo o negativo. Ej. "5" ó "-(5)"

  numero ::= NUMBER | MINUS LPAREN NUMBER RPAREN


//Una decisión lleva dos slots para comparar, un comparador que los compara y un número que dará el nuevo valor en caso de que la condición se cumpla

  decision  :== IF LPAREN slot comparador slot RPAREN THEN numero SEMI
```

```
// Dentro del cálculo que redondea debe haber sólo un Slot
redondeo :==  REDONDEAR LPAREN slot RPAREN SEMI
//Las siguientes producciones permiten definir en la misma gramática la precedencia de operadores. Una multiplicación o una división tienen más prioridad que una suma o una resta
expr_part ::=  expr  SEMI
expr     ::=  expr PLUS factor
          | expr MINUS factor
          | factor
factor   ::=  factor TIMES term
          | factor DIVIDE term
          | term
// Un término final de la gramática puede ser una expresión entre paréntesis, un número, un slot o un sumatorio.
term    ::=   LPAREN expr RPAREN
          | numero
          | slot
          | SUM1
          | SUM2
          | SUM3
          | SUM4
          | SUM5
          | SUM6
```

Puede parecer extraño reflejar los sumatorios como terminales. Esto se hace así porque dentro de ellos única y exclusivamente puede ir un atributo de la ontología (un slot) por lo que el sumatorio no es una producción que se pueda ampliar.

3.5.3 DESARROLLO DEL TRADUCTOR DE CÁLCULOS

El paquete "TranslatorPackage" contiene todas las clases necesarias para convertir los cálculos en **Maxima a reglas CLIPS**. Este paquete contiene las siguientes clases:

- **Traductor.class:** Clase encargada de pedir la carga de la información de la BBDD y crear los dos ficheros de texto donde se guardará la nueva ontología y las reglas.
- **GeneradorPredicciones.class:** Es la clase encargada de empezar a formar el archivo generador-predicciones.clp con las reglas de inicio. Su función es gestionar la lista de cálculos cargada de la BBDD para ir mandando traducirlas y encargarse también de la transición de un módulo a otro.
- **TraductorDecisiones.class:** Clase encargada de traducir los cálculos de tipo decisión a reglas CLIPS.
- **TraductorRedondeo.class:** Clase encargada de traducir los cálculos de tipo redondeo a reglas CLIPS.
- **TraductorImprimir.class:** Clase encargada de escribir la regla CLIPS que escribe los resultados en el fichero **Predicciones.out**. El método traducirImprimir() que contiene esta clase será llamado cada vez que se alcanza el final de un módulo.
- **TraductorSumatorioAcumuladoMercado.class:** Clase que traduce los sumatorios de un atributo para los cuatro períodos anteriores de todas las empresas.
- **TraductorSumatorioAcumuladoSector.class:** Exactamente igual que el anterior pero sólo para las empresas del sector.
- **TraductorSumatorioEmpresasMercado.class:** Clase que traduce los sumatorios de un atributo para todas las empresas dividido entre el número total de empresas.
- **TraductorSumatorioEmpresasSector:** Clase que traduce los sumatorios de un atributo para las empresas del sector dividido sólo entre el número de empresas del sector.
- **TraductorSumatorioSimpleMercado.class:** Clase que traduce sumatorios de atributos para todas las empresas del mercado.
- **TraductorSumatorioSimpleSector.class:** Igual que el anterior pero sólo para las empresas del mismo sector.
- **TraductorCalculos.class:** Esta clase se encarga de traducir el resto de cálculos que no se han tratado anteriormente. Aquí se incluyen los cálculos aritméticos que vienen definidos en la gramática.
- **Lexer.class:** Analizador léxico que se usa en la traducción.

- **Sym.class:** Clase que especifica los tipos de token que pueden ser reconocidos por el analizador léxico.
- **Parser.class:** Analizador sintáctico de los cálculos a traducir.

A continuación se describe sin entrar en demasiado detalle el proceso por el cual los cálculos almacenados en la base de datos en formato Maxima se convierten a reglas CLIPS en el fichero generador-predicciones. Recordar que si se quiere profundizar en el funcionamiento de la traducción se puede recurrir a la documentación JavaDoc de los apéndices o directamente al código fuente que está debidamente comentado.

Se comienza en la clase **Traductor.class creando los ficheros ontología-generica.clp y generador-predicciones.clp**. Aquí también se llama al método que carga toda la información de la BBDD. A continuación se pasa el control a la clase **GeneradorPredicciones.class.** En esta clase se escriben las reglas de inicio para instanciar las empresas, manejar el fichero de salida (Predicciones.out) y calcular el sector de cada empresa. Una vez hecho esto se comienza a tratar con los cálculos Maxima cargados de la BBDD. Se empieza por el primer cálculo del primer módulo y mientras existan reglas en el módulo actual se seguirá traduciendo. Cuando se cambia de módulo se llamará al método que escribe la regla de imprimir los resultados dentro de **TraductorImprimir.class.**

Dentro del bucle de mientras haya reglas se traduce, estos son los pasos que se suceden:

Si la regla es una decisión se llama al método traducirDecision() dentro de **TraducirDecisiones.class.** A este método, como a todos, se le pasa también la prioridad que debe asignarse a las reglas, para que al final sean las primeras reglas de la simulación aquellas con más prioridad y las últimas aquellas con menos prioridad.

Si el cálculo es un redondeo, se llama al método traducirRedondeo() de **TraductorRedondeo.class.** Por el contrario, sino es ni decisión ni redondeo se pasa al método traducirCalculo() de **TraductorCalculos.class.** Este método es bastante amplio, en él se comienza comprobando mediante el uso de expresiones regulares si en el cálculo existe alguno de los seis tipos de sumatorios. Si existe un sumatorio se llamará al método correspondiente de la clase que se encargue de ese tipo de sumatorio. En este método de sumatorio se generarán las reglas CLIPS

necesarias para obtener el resultado de este sumatorio y este valor se pasará al cálculo original donde el resultado del sumatorio será tratado.

Finalmente, llegado este punto, ya no se pueden tener ni decisiones, ni redondeos, ni sumatorios, sólo cálculos aritméticos simples compuestos de números y/o atributos acompañados de operadores. Estos atributos pueden ser slots de la ontología o slots de otros cálculos anteriores, es decir, en un cálculo se pueden usar resultados de otros cálculos.

Es a partir de aquí donde interviene la gramática del traductor. Una vez que ya se tiene una expresión aritmética con los operadores y los operandos sólo queda traducirlo a una regla CLIPS que sea ejecutable. Los cálculos que el usuario introduzca en el editor vendrán expresados en **notación infija** que es la comúnmente usada. Sin embargo, en CLIPS las operaciones vienen expresadas en **notación prefija** (también llamada notación polaca). Por ejemplo el cálculo DID(t) + DID(t-1) quedaría en CLIPS así: (+ DID(t) DID(t-1)).

El procedimiento a realizar es parecido al de la verificación, en primer lugar se pasa el cálculo por el analizador léxico (Lexer.class) y se obtienen los *tokens* pudiendo ser estos operadores (+, -, *, %) u operandos (números o slots). Estos *token* se pasarán al analizador sintáctico (Parser.class). El analizador sintáctico es creado mediante JCup al introducirle la especificación de la gramática que es esta:

expr_list ::= expr_list expr_part

| expr_part

expr_part ::= expr SEMI

expr ::= expr PLUS factor

| expr MINUS factor

| factor

factor ::= factor TIMES term

| factor DIVIDE term

| term

term ::= LPAREN expr RPAREN

| NUMBER

| ID (ID es un slot de un atributo de la ontología o de un cálculo anterior)

En esta gramática las precedencias ya quedan resueltas. **En el análisis sintáctico se escogerá la producción de la gramática que se pueda casar con el cálculo que se está traduciendo y se aplicará la acción que esté asociada a esa producción.**

Por ejemplo, si se está trabajando con una multiplicación 123 * 321. Este cálculo casará con la producción de la gramática factor ::= factor TIMES term. **La acción asociada a esta producción es la de escribir la operación en CLIPS con la forma "TIMES factor term" (operador operando operando).** De esta forma se obtendrá así el cálculo en notación prefija apto para ser introducido en la regla CLIPS.

Este proceso se repite hasta que no haya más cálculos que traducir. Cada vez que se cambia de módulo se escribe la regla que imprimirá los resultados en el fichero de Predicciones.out cuando se ejecute la simulación.

Otro apunte importante que añadir es el referente a la generación del fichero con la nueva ontología genérica. Este fichero contendrá las clases con los resultados de cada módulo. **Ontología-generica.clp se va creando a la vez que el fichero generador-predicciones.clp.** Antes de empezar a traducir cálculos se crea el fichero ontología-genérica.clp y se escriben las clases y slots que ya están en la base de datos. Posteriormente a medida que se van traduciendo los cálculos, por cada módulo del editor se crea una clase con el nombre ***"RESULTADOS + Nombre del Módulo"*** y por **cada cálculo se crea un slot** con el nombre del cálculo. De esta forma al llegar al final de la traducción se obtendrá el fichero con la nueva ontología genérica totalmente definido.

Para entender todo este proceso explicado, mejor hacerlo mediante un ejemplo descriptivo de la operación. Para ejemplificarlo se toma un cálculo de prueba cómo sería este:

$$CAL020 = \sum_{t-3\text{mercado}}^{t} \sum \; DID(t) + RTIPOEMP(t)$$

En formato Maxima sería tal que así:

```
sum(sum(DID(t),mercado,0,0),t,-3,t)+RTIPOEMP(t);
```

Como se puede ver este cálculo está compuesto de la suma de un sumatorio y un atributo cualquiera de la ontología. NOTA: El atributo RTIPOEMP(t) no está dentro del sumatorio sino que se suma al resultado final de este. En el cálculo Maxima este hecho se aprecia mejor.

Los pasos principales para traducir este cálculo a CLIPS son los siguientes:

1) Si es una decisión se llama al método traductorDecision() y se acaba el proceso. En este caso no lo es.

2) Si es una operación de redondeo se llama al método traducirRedondeo() y se acaba el proceso. Este cálculo tampoco es un redondeo.

3) Después se continúa con el método traducirCalculo() y se comprueba mediante el uso de expresiones regulares si existen sumatorios dentro del cálculo a traducir. En este caso hay un sumatorio del atributo DID de todas las empresas del mercado acumulado con el de los 3 períodos anteriores.

4) Lo que se hace es tomar el sumatorio y crear las reglas CLIPS que darán el resultado de ese sumatorio. El resultado de ese sumatorio se guardará en un slot temporal creado en la ontología. En este caso el nombre de ese slot sería CAL020A. Posteriormente se toma ese slot temporal y se añade al cálculo original por lo que ahora el cálculo quedaría así:

$$CAL020 = CAL020A + RTIPOEMP(t)$$

El único paso pendiente sería, mediante la gramática, transformar esta regla de notación infija a notación prefija que es la que usa CLIPS. En la regla CLIPS se crearía una sentencia parecida a esta:

```
(modify-instance ?res1 (ESTADOCAL 2) (CAL020 (+ ?cal020A ?rtipoemp)))
```

En esta sentencia se modifica el slot CAL020 de la empresa 1 (?res1) al que se le da el valor de la suma de ?cal020A + ?rtipoemp.

Finalmente si no hubiese más cálculos a traducir dentro del módulo en el que se está traduciendo se crearía la regla CLIPS dedicada a "imprimir" los resultados en el fichero "Predicciones.out".

Un punto a destacar es el referente a la gramática usada para traducir. Como se puede comprobar la gramática usada para verificar los cálculos es ligeramente diferente a la gramática usada para traducir. Esto es debido a que, como se ha visto, en el proceso de traducción los sumatorios se tratan con anterioridad antes de que pasen por la gramática haciendo que esta sea más corta pues debe tratar con menos elementos.

Además de los tres paquetes JAVA antes mencionados es necesaria la inclusión de dos librerías para el correcto funcionamiento de la traducción.

- **Mysql-coonector-java-5.1.17-bin.jar:** Driver encargado de la conexión con la BBDD de MySQL.
- **Java-cup-11a-runtime.jar:** Liberia necesaria para generar el parser sintáctico de nuestra gramática.

En la página siguiente se presenta el algoritmo de traducción explicado de forma gráfica lo que ayudará a comprender mejor el proceso.

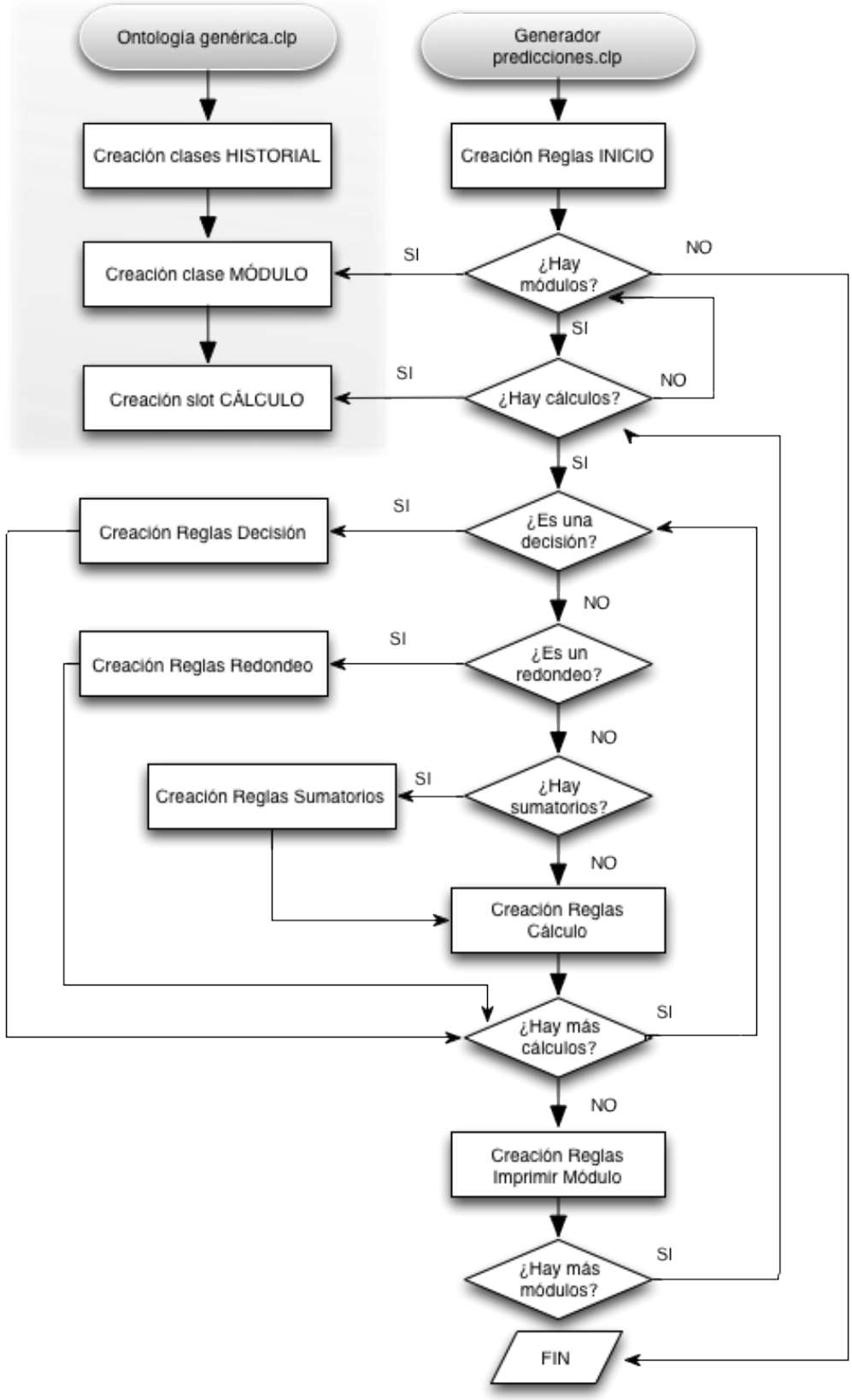


Figura 9: Algoritmo de traducción

4. INTEGRACIÓN DEL EDITOR Y EL TRADUCTOR

Como ya se comentó anteriormente, la comunicación entre ambas partes del proyecto general ha sido continua durante todo el proceso de diseño y desarrollo. **Existen restricciones a la hora de traducir los cálculos a CLIPS que deben ser tenidos en cuenta también en el editor.** Estas restricciones de integración entre el editor y el traductor pueden ser formulados en forma de requisitos y se listan a continuación.

- ✓ **Requisito de carga de la ontología genérica.** Cada vez que el usuario carga la ontología genérica se debe borrar la anterior guardada en la base de datos y ser sustituida por la nueva. Esta nueva ontología será puesta a disposición del usuario en el panel derecho del editor de cálculos.

- ✓ **Requisito de clase con información histórica.** Si los slots (atributos o cálculos) pertenecen a una clase que contiene un slot PERIODO (slot que indica el período de la información de la clase) significa que esa clase tiene información histórica almacenada, por lo tanto, todos los slots de esa clase deberán aparecer con las alternativas (t), (t-1), (t-2) y (t-3) para poder seleccionar de qué período se cogerá el atributo. Si una clase no tiene ese período discriminante no se mostrarán esas opciones.

- ✓ **Requisito para nombrar las reglas de la traducción.** Los nombres que se dan a las reglas CLIPS en la traducción es el mismo que se le da al cálculo del que derivan en el traductor. El nombre que toma el cálculo en el editor es automático y está compuesto por las tres primeras letras del nombre del módulo seguido por los tres dígitos del identificador de cálculo dentro de la base de datos. Por ejemplo, el cálculo 20 dentro de la BBDD que además está dentro del módulo Tesorería recibirá el nombre TES020.

- ✓ **Requisito para nombrar los módulos en el editor.** En el proceso de simulación cada módulo está identificado por una variable de tres letras que son las tres primeras letras del nombre del módulo. Para poder identificar sin ambigüedades cada módulo en el proceso

de simulación se requiere que ningún módulo tenga sus tres primeras letras iguales. Si fuese necesario tener varios módulos de Tesorería habría que añadir algún carácter diferenciador, por ejemplo, “1Tesorería”, “2Tesoreria”, etc.

- ✓ **Requisito de prioridad.** En la simulación CLIPS las reglas tienen asignada una prioridad de tal manera que las reglas con prioridad más alta se ejecutarán antes que las de menor prioridad. Debe ser posible en el editor cambiar el orden de los cálculos para que el usuario decida que cálculos se realizarán antes que otros.

- ✓ **Requisito de lenguaje de los cálculos:** Los cálculos guardados en la BBDD deben estar escritos en lenguaje Maxima. El traductor de reglas sólo puede leer, interpretar y traducir en ese formato.

Todo esto son ejemplos de cómo el análisis de CLIPS impone restricciones en el diseño del editor. Restricciones que surgen y son implementadas gracias a la coordinación entre ambos proyectos. Importante destacar que todo el desarrollo del sistema se iba haciendo sobre un entorno compartido, de tal forma que los avances eran vistos inmediatamente y así poder ir probando al mismo tiempo todos los avances facilitando aún más la integración.

Los tres paquetes de código JAVA creados en este proyecto con las dos librerías son añadidos a la carpeta **WEB-INF\classes** de la aplicación dentro del servidor Tomcat integrándose así con el resto del sistema. **Lo único que es necesario referenciar desde las páginas JSP de la aplicación del editor es el método comprobarRegla(regla) de la clase ComprobarRegla.class** que lanza una excepción en el caso de que la regla sea incorrecta (esta excepción es capturada en el JSP y su mensaje mostrado al usuario) y **el método traductor() dentro de la clase Traductor.class** que se encarga de generar la traducción y devolver el resultado al editor en forma de los dos archivos .clp que el usuario podrá ver o descargar para realizar su simulación en CLIPS.

5. VALIDACIÓN Y EVALUACIÓN

Las pruebas que se han realizado son muchas para verificar que el editor y por consiguiente el traductor funcionan correctamente. Se han realizado pruebas de verificación de cálculos (comprobando que no es posible guardar ningún cálculo si este ha sido escrito incorrectamente) y pruebas para comprobar que las reglas CLIPS generadas dan los resultados esperados.

En la siguiente hoja se detallan los cálculos que se han hecho dentro de una simulación de prueba. Algunos de estos cálculos estaban en la simulación que se proporcionó originalmente, otros se han añadido después para probar la versatilidad de la herramienta y correcto funcionamiento con los otros tipos de cálculos.

Se han realizado también infinidad de pruebas de integración del editor con el traductor para comprobar que no es posible guardar ningún cálculo que no sea correcto. Algunas de estas pruebas se adjuntan en los anexos del final ya que su extensión es demasiado grande para ser incluida aquí.

NOTA: Todas las simulaciones CLIPS se han realizado en un entorno Linux mediante la aplicación de CLIPS de la distribución Ubuntu 11.0.

5.1 COMPROBACIÓN DE LA SIMULACIÓN

CÁLCULOS

Los cálculos se encuentran escritos en formato Maxima. Al cargarlos en el editor se pueden ver correctamente formados en un lenguaje matemático más sencillo de interpretar.

CAL000 sum(sum(DID(t),sector,0,0),t,-3,t);

$$\sum_{t-3}^{t}\sum_{\text{sector}}\text{DID}(t)$$

CAL120 sum(sum(DID(t),mercado,0,0),t,-3,t);

$$\sum_{t-3}^{t}\sum_{\text{mercado}}\text{DID}(t)$$

CAL121 (DID(t)-DID(t-1)/DID(t-1))*100;

$$\left(\frac{\text{DID}\times(t)-\text{DID}\times(t-1)}{\text{DID}\times(t-1)}\right)\times 100$$

CAL122 sum(DPUBLICIDA(t-1),sector,0,0)/n;

$$\frac{\sum_{\text{sector}}\text{DPUBLICIDA}(t-1)}{n}$$

CAL123 sum(sum(DPUBLICIDA(t),mercado,0,0),t,-3,t);

$$\sum_{t-3}^{t}\sum_{\text{mercado}}\text{DPUBLICIDA}(t)$$

CAL124 RTIPOEMP(t-1)-RTIPOEMP(t-2)/100;

$$\text{RTIPOEMP}(t-1)-\frac{\text{RTIPOEMP}(t-2)}{100}$$

CAL125 if (RCAPINST(t-1)=RCAPINST(t-1))then123;

CAL126 123.5+-(1.5);

CAL127 CAL120(t)/CAL121(t);

CAL128 REDONDEAR(CAL127(t));

RESULTADOS

Tras la simulación (todos los cálculos se encuentran dentro del módulo de Calidad) los resultados que se obtienen son estos:

Predicciones para el período t= 1

Resultados CALIDAD para Equipo1

CalculoCAL000: 1340154.82075288

CalculoCAL120: 4507488.61093681

CalculoCAL121: 15724179.6874654

CalculoCAL122: 53409.9423441895

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.28665969866332

Calculo CAL128: 0

Resultados CALIDAD para Equipo2

CalculoCAL000: 3167333.79018393

CalculoCAL120: 4507488.61093681

CalculoCAL121: 34497994.7926658

CalculoCAL122: 83340.3451432213

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.13065943797681

Calculo CAL128: 0

Resultados CALIDAD para Equipo3

CalculoCAL000: 1340154.82075288

CalculoCAL120: 4507488.61093681

CalculoCAL121: 18150465.5529357

CalculoCAL122: 53409.9423441895

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.2483401099432

Calculo CAL128: 0

Resultados CALIDAD para Equipo4

CalculoCAL000: 3167333.79018393

CalculoCAL120: 4507488.61093681

CalculoCAL121: 27045444.6980829

CalculoCAL122: 83340.3451432213

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.16666350512093

Calculo CAL128: 0

Resultados CALIDAD para Equipo5

CalculoCAL000: 3167333.79018393

CalculoCAL120: 4507488.61093681

CalculoCAL121: 25843420.4892793

CalculoCAL122: 83340.3451432213

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.17441532605200

Calculo CAL128: 0

Resultados CALIDAD para Equipo6

CalculoCAL000: 1340154.82075288

CalculoCAL120: 4507488.61093681

CalculoCAL121: 9015081.56602765

CalculoCAL122: 53409.9423441895

Calculo CAL123: 1764769.8875864

Calculo CAL124: 7.92

Calculo CAL125: 123

Calculo CAL126: 122.0

CalculoCAL127: 0.49999421280033

Calculo CAL128: 0

6. CONCLUSIONES

La realización de este proyecto Fin de Carrera ha permitido comprender **cómo es desarrollar una solución informática de principio a fin** partiendo de la problemática inicial a resolver, analizando la situación y el estado del simulador para determinar cómo mejorar el sistema y hacerlo más versátil para todo tipo de usuarios.

Después de realizar distintas reuniones iniciales para conocer el estado de del sistema **hubo un intenso proceso de análisis** y acercamiento para conocer el funcionamiento interno del simulador. Se ha comprendido así lo importante que es obtener un conjunto de requisitos que permitan definir el camino a seguir.

Tras este acercamiento inicial se continuó con el proceso de diseño de la solución. Cómo se iba a guardar la información que el sistema gestionaba, cómo se iba a cambiar el flujo de ejecución del simulador para adaptarlo a los nuevos requerimientos y hacer pruebas para comprobar que este cambio funcionaba.

A la vez se empezaron a estudiar las tecnologías que podrían ayudar a implementar la solución. Una de las partes que más problemática planteaba era el **editor de reglas** que permitía arrastrar y soltar elementos del panel de componentes al área de edición. Aunque el editor forma parte del otro subproyecto y no se habla de estas tecnologías en este documento merece la pena destacarlo ya que supuso un reto importante para el avance en paralelo de los dos proyectos. En un principio se estudió el uso de HTML5 , aún en borrador, y otras soluciones, hasta que finalmente se encontró DragMath (http://www.dragmath.bham.ac.uk/), un editor de ecuaciones de código libre que se pudo modificar para adaptar a los requisitos de ambos proyectos, viendo así lo importante que es la comunidad de desarrolladores para implementar cada vez herramientas más eficaces y potentes.

Después de haber realizado las modificaciones en el editor de DragMath por un lado y haber testeado que las nuevas reglas CLIPS diseñadas daban como resultado los mismos valores que antes por otro, se continuó con el desarrollo. Por un lado se avanzaba creando todo el entorno que rodea al editor y compositor de cálculos y por otro implementando en Java la funcionalidad necesaria para tomar los cálculos y traducirlos a reglas CLIPS. Todo ello contando con la base de datos MySQL como interfaz entre ambos. Se ha visto así también **cuan importantes son las reuniones de coordinación y sincronización** entre ambas partes, a veces por videoconferencia y otras presenciales, para poder comunicarse mutuamente los avances y comprobar que los desarrollos en paralelo de cada proyecto no "chocan" en alguna restricción.

Uno de los problemas más graves que se encontraron durante el desarrollo fue el dar marcha atrás con el lenguaje que se usaba de intermediario entre el editor y el traductor. La base de datos MySQL guarda información de cada cálculo que es modificado o introducido por el usuario. Se guarda su identificador, su nombre, el módulo al que pertenece y el cuerpo del cálculo en sí mismo. Es el cuerpo del cálculo el que se parsea para traducir a CLIPS, por eso es importante que el **lenguaje que se usa para contener el cálculo sea suficientemente potente** para albergar toda la información del cálculo y sus restricciones (sector, mercado, etc.) En un primer momento se consideró trabajar con el lenguaje LaTeX, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. Este permitía hacer todo ello, incluso se empezó a trabajar con las decisiones escritas en LaTeX para traducir a CLIPS. No fue hasta en una de las múltiples sesiones de reunión cuando se descubrió que aunque la composición de los cálculos se podía hacer correctamente no pasaba lo mismo con la carga de cálculos. Viendo el resto de lenguajes alternativos se comprobó que con Maxima sí que permitía grabar y cargar de la base de datos sin problemas.

Este es un ejemplo que muestra lo importante que es la sincronización y la planificación en el desarrollo de un proyecto, más aún si el equipo fuera grande y el rol de analista, diseñador y programador fuera interpretado por diferentes personas. Estos problemas son a veces ineludibles, y **sólo el tiempo y la experiencia pueden evitarlos.**

Como conclusión final se puede afirmar que **se han cubierto los objetivos planteados en esta parte del proyecto.** Por un lado se ha proporcionado al sistema la funcionalidad para verificar que todos los cálculos introducidos en el mismo son correctos y que son aptos para la traducción a reglas CLIPS.

Por otro lado **la generación de los archivos CLIPS** (ontología-genérica.clp y generador-predicciones.clp) **es totalmente satisfactoria.** Una vez que el usuario acaba de añadir y editar los cálculos y escoge la opción de traducción para descargar ambos ficheros, ya sólo hay que juntarlos con el resto de archivos CLIPS de la simulación (vistos en al sección 3.3.1) y proceder a la ejecución. Los nuevos valores de la simulación son los que aparecen en el fichero de resultados predicciones.out. Se ha proporcionado así una forma fácil y rápida de cambiar los cálculos que controlan y dirigen el flujo de ejecución de la simulación sin tener que acceder directamente a modificar las reglas CLIPS, tarea complicada y engorrosa que mediante esta solución es evitada en su totalidad.

7. TRABAJOS FUTUROS

Una vez acabada la implementación del proyecto y haber podido probar el funcionamiento del editor **se puede afirmar con seguridad que este cumple con el propósito con el que fue creado**. El usuario que lo utilice puede crear, eliminar y editar cálculos, así como clasificarlos dentro de sus respectivos módulos y ordenarlos por prioridad. También es posible cargar la ontología que desee según la información histórica de la que se disponga. Y por último puede traducir los cálculos a CLIPS y obtener los ficheros para poder ejecutar la simulación dentro del entorno de simulación.

Sin embargo **son varias las ampliaciones que se pueden realizar** al proyecto como posibles trabajos futuros. Quizá el que más destaque sea el de poder **cambiar el atributo discriminante** usado para determinar si una empresa pertenece al sector alto, medio o bajo. Por defecto, en este sistema, el atributo que se usa es el **RCALI, que pertenece a la clase Historial Área Comercial** dentro de la ontología. El tener un apartado dentro de la aplicación para poder cambiar dicho atributo sería una manera de flexibilizar y aportar mayor versatilidad a la solución así como una mayor capacidad de maniobra al usuario y/o administrador.

Otra posible mejora consiste en poder **cambiar el número de empresas** que participan en la simulación. Por defecto **no hay ninguna forma de "decir" al editor cuantas empresas componen la simulación** por lo que los ficheros CLIPS que se devuelven siempre suponen que son seis las empresas que van a participar en la simulación. Una forma de introducir este dato en el editor permitiría traducir y ejecutar los cálculos para un mayor o menor número de empresas.

Una última mejora de accesibilidad sería la de poder acceder al editor **desde cualquier plataforma y dispositivo móvil**. Debido que la solución implementada usa la tecnología Java y esta no puede ser usada en la mayoría de navegadores móviles sería necesario implementar la solución de nuevo usando otra tecnología compatible o desarrollarlo de forma específica para la plataforma que se desee.

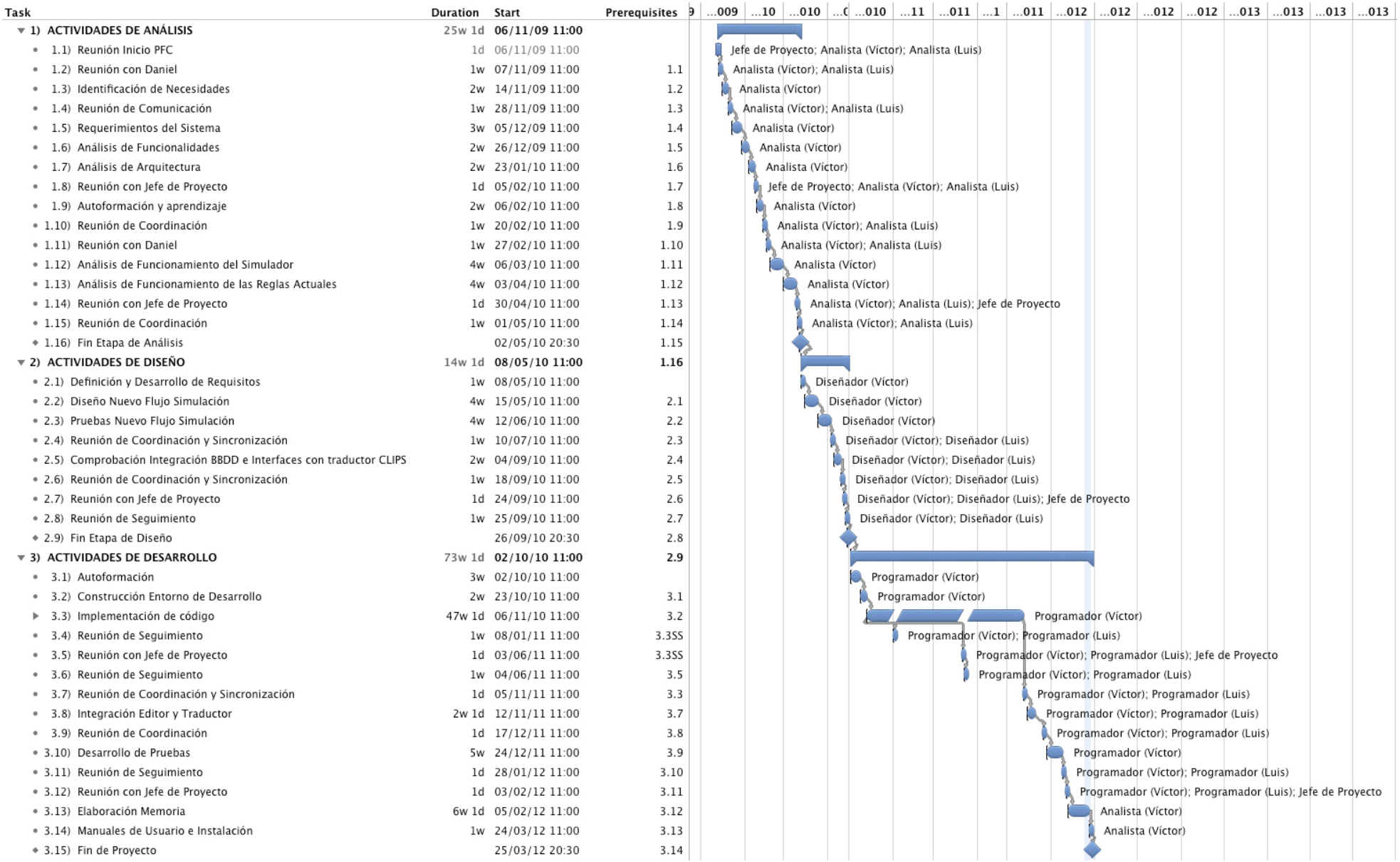
8. PLANIFICACIÓN Y PRESUPUESTO

La planificación es algo que siempre hay que tener presente a la hora de acometer un proyecto, sobre todo cuando este se alarga en el tiempo y participan muchas personas con distintas metodologías y formas de trabajar. En este caso, las personas que han colaborado en este proyecto se conocían de haber trabajado durante asignaturas de la carrera por lo que la forma de trabajar era conocida para ambos. Ambos hemos tenido que compaginar la realización de este proyecto con nuestro trabajo además de con otras asignaturas pendientes, lo que unido a los imprevistos que se comentaban en el apartado anterior ha hecho que todo se demorase más de lo deseado.

En esta sección se muestra la planificación y desarrollo del proyecto a lo largo del tiempo para poder apreciar las tareas que se han acometido hasta llegar a la solución final. Las tareas que componen la planificación son las pertenecientes al desarrollo de este proyecto aunque lógicamente en muchas de ellas también se requiere la participación del autor del proyecto análogo a este.

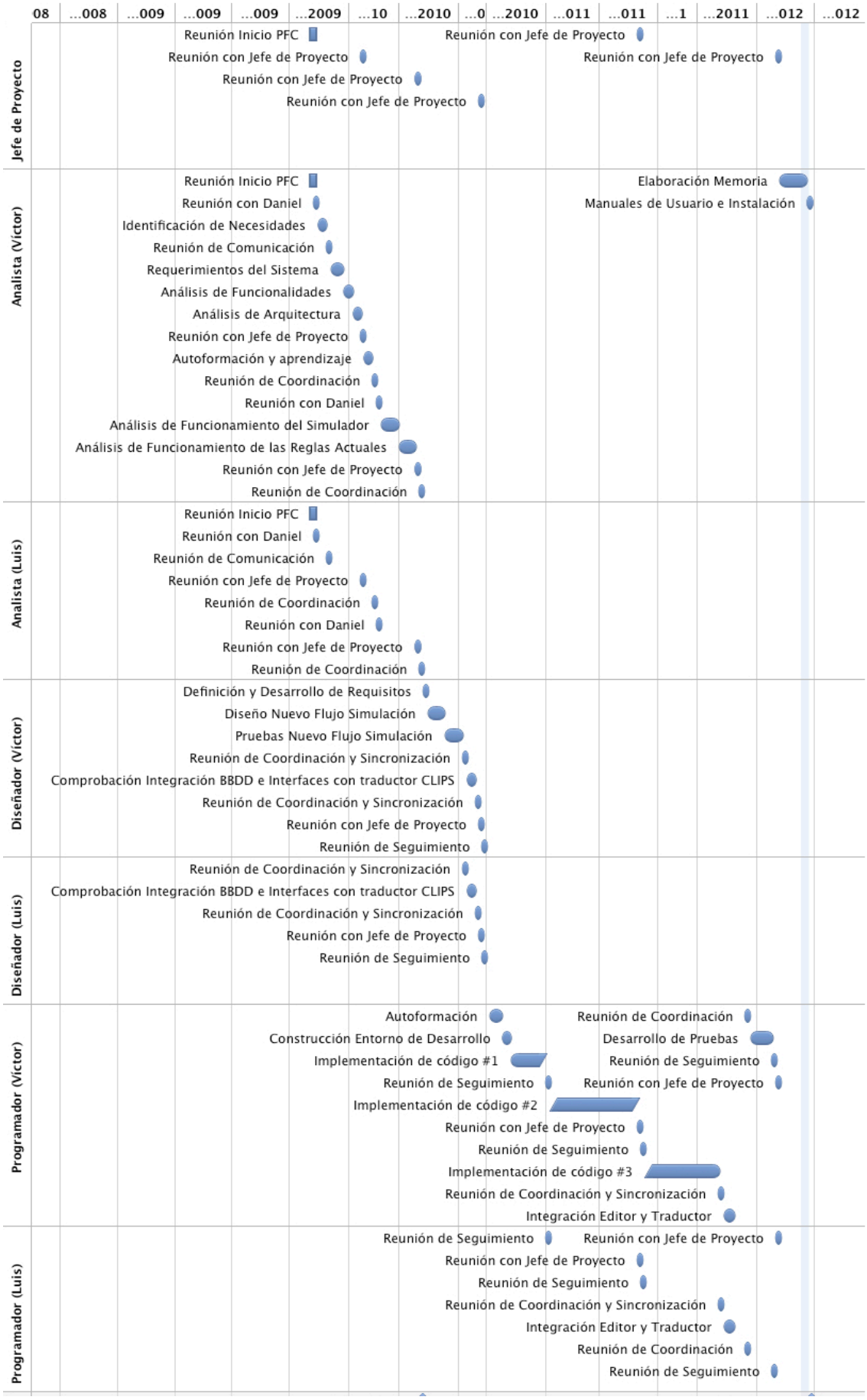
Aunque el proyecto general ha sido desarrollado sólo por dos personas, a la hora de especificar los recursos los autores tomarán diferentes roles. El tutor Fernando Fernández hará el papel de Jefe de Proyecto y participará en las reuniones importantes de seguimiento. Por otra parte Luis Echevarría y yo interpretaremos los papeles de analista, diseñador y programador según corresponda.

Diagrama de Gantt



Task	Duration	Start	Prerequisites
1) ACTIVIDADES DE ANÁLISIS	25w 1d	06/11/09 11:00	
1.1) Reunión Inicio PFC	1d	06/11/09 11:00	
1.2) Reunión con Daniel	1w	07/11/09 11:00	1.1
1.3) Identificación de Necesidades	2w	14/11/09 11:00	1.2
1.4) Reunión de Comunicación	1w	28/11/09 11:00	1.3
1.5) Requerimientos del Sistema	3w	05/12/09 11:00	1.4
1.6) Análisis de Funcionalidades	2w	26/12/09 11:00	1.5
1.7) Análisis de Arquitectura	2w	23/01/10 11:00	1.6
1.8) Reunión con Jefe de Proyecto	1d	05/02/10 11:00	1.7
1.9) Autoformación y aprendizaje	2w	06/02/10 11:00	1.8
1.10) Reunión de Coordinación	1w	20/02/10 11:00	1.9
1.11) Reunión con Daniel	1w	27/02/10 11:00	1.10
1.12) Análisis de Funcionamiento del Simulador	4w	06/03/10 11:00	1.11
1.13) Análisis de Funcionamiento de las Reglas Actuales	4w	03/04/10 11:00	1.12
1.14) Reunión con Jefe de Proyecto	1d	30/04/10 11:00	1.13
1.15) Reunión de Coordinación	1w	01/05/10 11:00	1.14
1.16) Fin Etapa de Análisis		02/05/10 20:30	1.15
2) ACTIVIDADES DE DISEÑO	14w 1d	08/05/10 11:00	1.16
2.1) Definición y Desarrollo de Requisitos	1w	08/05/10 11:00	
2.2) Diseño Nuevo Flujo Simulación	4w	15/05/10 11:00	2.1
2.3) Pruebas Nuevo Flujo Simulación	4w	12/06/10 11:00	2.2
2.4) Reunión de Coordinación y Sincronización	1w	10/07/10 11:00	2.3
2.5) Comprobación Integración BBDD e Interfaces con traductor CLIPS	2w	04/09/10 11:00	2.4
2.6) Reunión de Coordinación y Sincronización	1w	18/09/10 11:00	2.5
2.7) Reunión con Jefe de Proyecto	1d	24/09/10 11:00	2.6
2.8) Reunión de Seguimiento	1w	25/09/10 11:00	2.7
2.9) Fin Etapa de Diseño		26/09/10 20:30	2.8
3) ACTIVIDADES DE DESARROLLO	73w 1d	02/10/10 11:00	2.9
3.1) Autoformación	3w	02/10/10 11:00	
3.2) Construcción Entorno de Desarrollo	2w	23/10/10 11:00	3.1
3.3) Implementación de código	47w 1d	06/11/10 11:00	3.2
3.4) Reunión de Seguimiento	1w	08/01/11 11:00	3.3SS
3.5) Reunión con Jefe de Proyecto	1d	03/06/11 11:00	3.3SS
3.6) Reunión de Seguimiento	1w	04/06/11 11:00	3.5
3.7) Reunión de Coordinación y Sincronización	1d	05/11/11 11:00	3.3
3.8) Integración Editor y Traductor	2w 1d	12/11/11 11:00	3.7
3.9) Reunión de Coordinación	1d	17/12/11 11:00	3.8
3.10) Desarrollo de Pruebas	5w	24/12/11 11:00	3.9
3.11) Reunión de Seguimiento	1d	28/01/12 11:00	3.10
3.12) Reunión con Jefe de Proyecto	1d	03/02/12 11:00	3.11
3.13) Elaboración Memoria	6w 1d	05/02/12 11:00	3.12
3.14) Manuales de Usuario e Instalación	1w	24/03/12 11:00	3.13
3.15) Fin de Proyecto		25/03/12 20:30	3.14

Timeline de Recursos



Presupuesto por recurso humano

Name	Start Date	End Date	Duration	% Complete	Assignment Cost
Jefe de Proyecto	**06/11/09 11:00**	**04/02/12 20:30**	**3w**	**100%**	**€ 300,00**
Reunión Inicio PFC	06/11/09 11:00	06/11/09 19:00	1d	100%	€ 50,00
Reunión con Jefe de Proyecto	05/02/10 11:00	05/02/10 19:00	1d	100%	€ 50,00
Reunión con Jefe de Proyecto	30/04/10 11:00	30/04/10 19:00	1d	100%	€ 50,00
Reunión con Jefe de Proyecto	24/09/10 11:00	24/09/10 19:00	1d	100%	€ 50,00
Reunión con Jefe de Proyecto	03/06/11 11:00	03/06/11 19:00	1d	100%	€ 50,00
Reunión con Jefe de Proyecto	03/02/12 11:00	04/02/12 20:30	1d	100%	€ 50,00
Analista (Víctor)	**06/11/09 11:00**	**25/03/12 20:30**	**33w**	**100%**	**€ 15.840,00**
Reunión Inicio PFC	06/11/09 11:00	06/11/09 19:00	1d	100%	€ 240,00
Reunión con Daniel	07/11/09 11:00	08/11/09 20:30	1w	100%	€ 480,00
Identificación de Necesidades	14/11/09 11:00	22/11/09 20:30	2w	100%	€ 960,00
Reunión de Comunicación	28/11/09 11:00	29/11/09 20:30	1w	100%	€ 480,00
Requerimientos del Sistema	05/12/09 11:00	20/12/09 20:30	3w	100%	€ 1.440,00
Análisis de Funcionalidades	26/12/09 11:00	17/01/10 20:30	2w	100%	€ 960,00
Análisis de Arquitectura	23/01/10 11:00	31/01/10 20:30	2w	100%	€ 960,00
Reunión con Jefe de Proyecto	05/02/10 11:00	05/02/10 19:00	1d	100%	€ 240,00
Autoformación y aprendizaje	06/02/10 11:00	14/02/10 20:30	2w	100%	€ 960,00
Reunión de Coordinación	20/02/10 11:00	21/02/10 20:30	1w	100%	€ 480,00
Reunión con Daniel	27/02/10 11:00	28/02/10 20:30	1w	100%	€ 480,00
Análisis de Funcionamiento del Simulador	06/03/10 11:00	28/03/10 20:30	4w	100%	€ 1.920,00
Análisis de Funcionamiento de las Reglas Actuales	03/04/10 11:00	25/04/10 20:30	4w	100%	€ 1.920,00
Reunión con Jefe de Proyecto	30/04/10 11:00	30/04/10 19:00	1d	100%	€ 240,00
Reunión de Coordinación	01/05/10 11:00	02/05/10 20:30	1w	100%	€ 480,00
Elaboración Memoria	05/02/12 11:00	18/03/12 20:30	6w 1d	100%	€ 3.120,00
Manuales de Usuario e Instalación	24/03/12 11:00	25/03/12 20:30	1w	100%	€ 480,00
Analista (Luis)	**06/11/09 11:00**	**02/05/10 20:30**	**6w 1d**	**100%**	**€ 3.120,00**
Reunión Inicio PFC	06/11/09 11:00	06/11/09 19:00	1d	100%	€ 240,00
Reunión con Daniel	07/11/09 11:00	08/11/09 20:30	1w	100%	€ 480,00
Reunión de Comunicación	28/11/09 11:00	29/11/09 20:30	1w	100%	€ 480,00
Reunión con Jefe de Proyecto	05/02/10 11:00	05/02/10 19:00	1d	100%	€ 240,00
Reunión de Coordinación	20/02/10 11:00	21/02/10 20:30	1w	100%	€ 480,00
Reunión con Daniel	27/02/10 11:00	28/02/10 20:30	1w	100%	€ 480,00
Reunión con Jefe de Proyecto	30/04/10 11:00	30/04/10 19:00	1d	100%	€ 240,00
Reunión de Coordinación	01/05/10 11:00	02/05/10 20:30	1w	100%	€ 480,00
Diseñador (Víctor)	**08/05/10 11:00**	**26/09/10 20:30**	**14w 1d**	**100%**	**€ 5.800,00**
Definición y Desarrollo de Requisitos	08/05/10 11:00	09/05/10 20:30	1w	100%	€ 400,00
Diseño Nuevo Flujo Simulación	15/05/10 11:00	06/06/10 20:30	4w	100%	€ 1.600,00
Pruebas Nuevo Flujo Simulación	12/06/10 11:00	04/07/10 20:30	4w	100%	€ 1.600,00
Reunión de Coordinación y Sincronización	10/07/10 11:00	11/07/10 20:30	1w	100%	€ 400,00
Comprobación Integración BBDD e Interfaces con traductor CLIPS	04/09/10 11:00	12/09/10 20:30	2w	100%	€ 800,00
Reunión de Coordinación y Sincronización	18/09/10 11:00	19/09/10 20:30	1w	100%	€ 400,00
Reunión con Jefe de Proyecto	24/09/10 11:00	24/09/10 19:00	1d	100%	€ 200,00
Reunión de Seguimiento	25/09/10 11:00	26/09/10 20:30	1w	100%	€ 400,00
Diseñador (Luis)	**10/07/10 11:00**	**26/09/10 20:30**	**5w 1d**	**100%**	**€ 2.200,00**
Reunión de Coordinación y Sincronización	10/07/10 11:00	11/07/10 20:30	1w	100%	€ 400,00
Comprobación Integración BBDD e Interfaces con traductor CLIPS	04/09/10 11:00	12/09/10 20:30	2w	100%	€ 800,00
Reunión de Coordinación y Sincronización	18/09/10 11:00	19/09/10 20:30	1w	100%	€ 400,00
Reunión con Jefe de Proyecto	24/09/10 11:00	24/09/10 19:00	1d	100%	€ 200,00
Reunión de Seguimiento	25/09/10 11:00	26/09/10 20:30	1w	100%	€ 400,00
Programador (Víctor)	**02/10/10 11:00**	**04/02/12 20:30**	**62w**	**100%**	**€ 19.840,00**
Autoformación	02/10/10 11:00	17/10/10 20:30	3w	100%	€ 960,00
Construcción Entorno de Desarrollo	23/10/10 11:00	31/10/10 20:30	2w	100%	€ 640,00
Implementación de código #1	06/11/10 11:00	26/12/10 20:30	8w	100%	€ 2.560,00
Reunión de Seguimiento	08/01/11 11:00	09/01/11 20:30	1w	100%	€ 320,00
Implementación de código #2	15/01/11 11:00	29/05/11 20:30	20w	100%	€ 6.400,00
Reunión con Jefe de Proyecto	03/06/11 11:00	03/06/11 19:00	1d	100%	€ 160,00
Reunión de Seguimiento	04/06/11 11:00	05/06/11 20:30	1w	100%	€ 320,00
Implementación de código #3	11/06/11 11:00	30/10/11 20:30	17w	100%	€ 5.440,00
Reunión de Coordinación y Sincronización	05/11/11 11:00	05/11/11 20:30	1d	100%	€ 160,00
Integración Editor y Traductor	12/11/11 11:00	26/11/11 20:30	2w 1d	100%	€ 800,00
Reunión de Coordinación	17/12/11 11:00	17/12/11 20:30	1d	100%	€ 160,00
Desarrollo de Pruebas	24/12/11 11:00	22/01/12 20:30	5w	100%	€ 1.600,00
Reunión de Seguimiento	28/01/12 11:00	28/01/12 20:30	1d	100%	€ 160,00
Reunión con Jefe de Proyecto	03/02/12 11:00	04/02/12 20:30	1d	100%	€ 160,00
Programador (Luis)	**08/01/11 11:00**	**04/02/12 20:30**	**7w 1d**	**100%**	**€ 2.400,00**
Reunión de Seguimiento	08/01/11 11:00	09/01/11 20:30	1w	100%	€ 320,00
Reunión con Jefe de Proyecto	03/06/11 11:00	03/06/11 19:00	1d	100%	€ 160,00
Reunión de Seguimiento	04/06/11 11:00	05/06/11 20:30	1w	100%	€ 320,00
Reunión de Coordinación y Sincronización	05/11/11 11:00	05/11/11 20:30	1d	100%	€ 160,00
Integración Editor y Traductor	12/11/11 11:00	26/11/11 20:30	2w 1d	100%	€ 800,00
Reunión de Coordinación	17/12/11 11:00	17/12/11 20:30	1d	100%	€ 160,00
Reunión de Seguimiento	28/01/12 11:00	28/01/12 20:30	1d	100%	€ 160,00
Reunión con Jefe de Proyecto	03/02/12 11:00	04/02/12 20:30	1w	100%	€ 320,00

Presupuesto Material (Hardware y Software)

HARDWARE

• MacBook Pro 2009	1290,00 €
• MacBook Air 2011	1174,00 €

SOFTWARE

• SO Mac OSX 10.6 Snow Leopard	Incluido en equipo
• SO Mac OSX 10.7 Lion	Incluido en equipo
• SW de Virtualización Parallels Desktop 6.0	69,00 €
• SO virtualizado Ubuntu Linux 11	0,00 €
• SO virtualizado Windows 7 (MSDN UC3M)	0,00 €
• Microsoft Office for Mac	109,00 €
• Omniplan 2.0.3 (Versión de prueba)	0,00 €
• Keynote	15,99 €
• CLIPS (Linux)	0,00 €
• IDE Netbeans 7.0 for Mac	0,00 €
• MAMP 2.0.1 (Mac, Apache, MySQL, PHP)	0,00 €

PRESUPUESTO MATERIAL TOTAL	2.657,99 €
PRESUPUESTO RECURSOS HUMANOS	49.500,00 €
PRESUPUESTO FINAL	52.157,99 €

9. ANEXOS

En esta última sección de la memoria se adjunta información complementaria, que aún no siendo imprescindible para entender el proceso de creación de la herramienta aporta información muy útil sobre ella.

9.1 MANUAL DE USO DE LA APLICACIÓN

El objetivo de este manual es el de proporcionar una visión rápida y sencilla del manejo del editor de cálculos. Cómo cargar una ontología para usar su información, cómo crear, editar y eliminar los cálculos que componen el motor de razonamiento del simulador y cómo extraerlos para poder usarlos en las futuras simulaciones a realizar.

INDICE

Pantalla de Autenticación

Figura 10: Pantalla de entrada

En la pantalla de autenticación se muestran dos campos para introducir las credenciales que el administrador del sistema haya proporcionado. Una vez introducidas sólo hay que pulsar el botón de *Enviar*.

Pantalla de Inicio

Tras el proceso de autenticación aparecerá la pantalla principal donde se muestra un esquema con los pasos necesarios para completar un ciclo de uso de la aplicación (cargar Ontología, crear y editar cálculos, descarga de ficheros). Es posible ir a uno de los pasos directamente si los otros ya se hicieron con anterioridad.

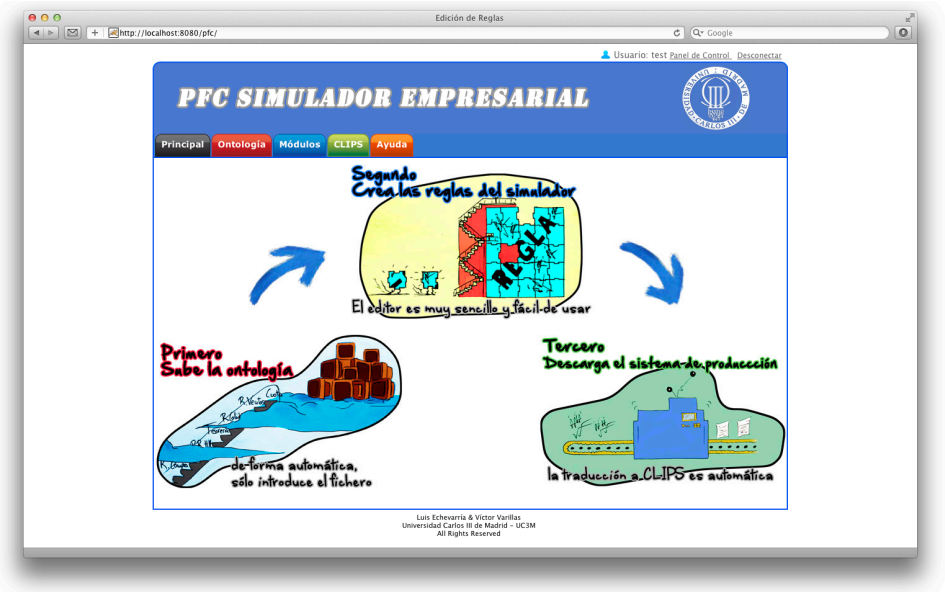


Figura 11: Pantalla inicial del sistema

Carga de la Ontología

El primer paso para poder crear y editar cálculos es cargar la ontología genérica con la información de las clases y slots que representan el conjunto de datos históricos del entorno y las empresas del mercado a lo largo del tiempo.

En la pestaña *Ontología* es donde se puede hacer esto.

Figura 12: Carga de la Ontología

Sólo es necesario seleccionar el fichero y subirlo en formato CLIPS (extensión .clp). De esta manera ya aparecerá en el panel derecho del editor de cálculos que se ve en la siguiente sección. Si se produce algún error aparecerá la pantalla que sigue en la siguiente hoja.

Figura 13: Pantalla de error

Creación y edición de cálculos

Bajo la pestaña de *Módulos* se encuentra el componente de la aplicación que permite trabajar con los módulos y cálculos del simulador. La primera pantalla que aparece es la que muestra todos los módulos que ayudan a separar los cálculos en diferentes categorías.

Figura 14: Pantalla de vista de módulos

Si se quiere borrar un módulo junto con los cálculos que este contiene sólo es necesario hacer click en el aspa roja que aparece al lado del nombre del módulo. Si se quiere editar el nombre del módulo se debe seleccionar este icono

Aparece así esta pantalla para introducir el nuevo nombre. Tras pulsar sobre *Modificar el módulo* se indicará que los cambios se han completado satisfactoriamente.

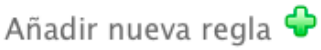
Figura 15: Pantalla edición de módulo

Para ver los cálculos que pertenecen a ese módulo sólo se debe hacer *click* sobre el nombre del módulo. Se ven así todos los cálculos escritos en lenguaje Maxima.

Figura 16: Pantalla de cálculos

Se puede también cambiar el orden en el que se ejecutarán los cálculos, teniendo siempre cuidado de mantener las precedencias, por ejemplo, no poner a ejecutar un cálculo que necesita de un cálculo anterior antes que ese cálculo del que se necesita saber su resultado. Para cambiar el orden sólo hay que usar las flechas que aparecen a la izquierda del nombre.

Igualmente que para los módulos, si se desea borrar una regla sólo se ha de hacer *click* sobre el aspa roja. Si se quiere editar la regla pulsar en el icono del papel con el lápiz. Si se prefiere añadir una nueva regla se ha de escoger la opción de Añadir nueva regla.



Tanto si se escoge editar un cálculo como crear uno nuevo aparecerá la pantalla del editor. Si se está editando aparecerá la regla que estaba creada pero ya no en formato Maxima, sino en lenguaje matemático.

Cuando se va a entrar en el editor primero se ha de conceder los permisos para que se ejecute el *applet* pulsando en *Aceptar*.

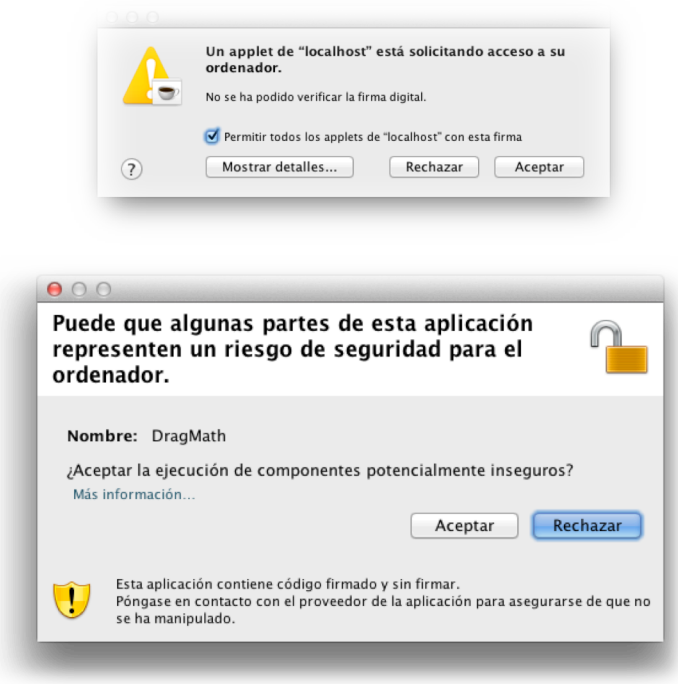


Figura 17: Panel de Advertencia

Cuando se han dado los permisos aparecerá ya la pantalla entera del editor. El nombre del cálculo se asigna automáticamente con las tres primeras letras del nombre del módulo y el siguiente identificador de cálculo que corresponde en la base de datos.

En el panel derecho del editor se pueden ver todas las clases con sus respectivos *slots* cargados de la ontología genérica. Si una clase de la ontología tiene un atributo PERIODO significa que existe información histórica de sus atributos (por ejemplo, el atributo DPRECIOEMP ha tomado distintos valores según el período), por lo que aparecerán todos los atributos de esa clase con un distintivo (t), (t-1), (t-2) y (t-3) para seleccionar el valor que ha tomado ese atributo en un determinado período.

Para crear un nuevo cálculo existen dos opciones: teclear directamente el nombre exacto de los atributos o coger y arrastrar los operadores de la parte superior y los atributos de la parte derecha.

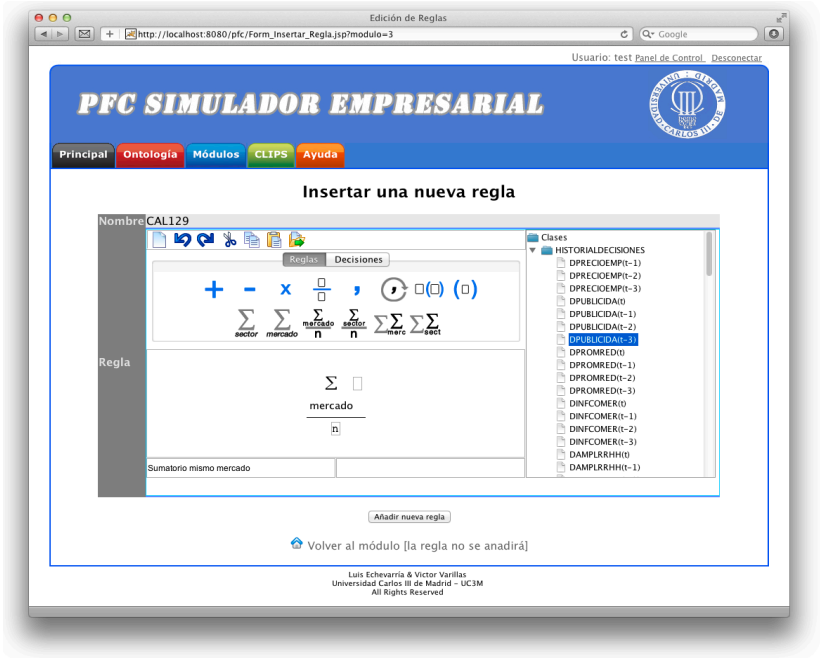


Figura 18: Panel de edición de regla

Si el operando que se quiere introducir es un número en vez de un atributo ha de escribirse obligatoriamente por teclado. Cuando lo que se quiere crear es una decisión, es decir, asignar un valor determinado al slot del resultado cuando se cumple una condición, se debe recurrir a la pestaña *Decisiones* que aparece junto a la pestaña *Reglas*, justo encima de los operadores. En la condición se pueden usar los comparadores "=", ">=" y "<=" y sólo se podrán comparar atributos de la ontología.

Una vez creado o editado el cálculo sólo hay que pulsar el botón de *Añadir nueva regla.*

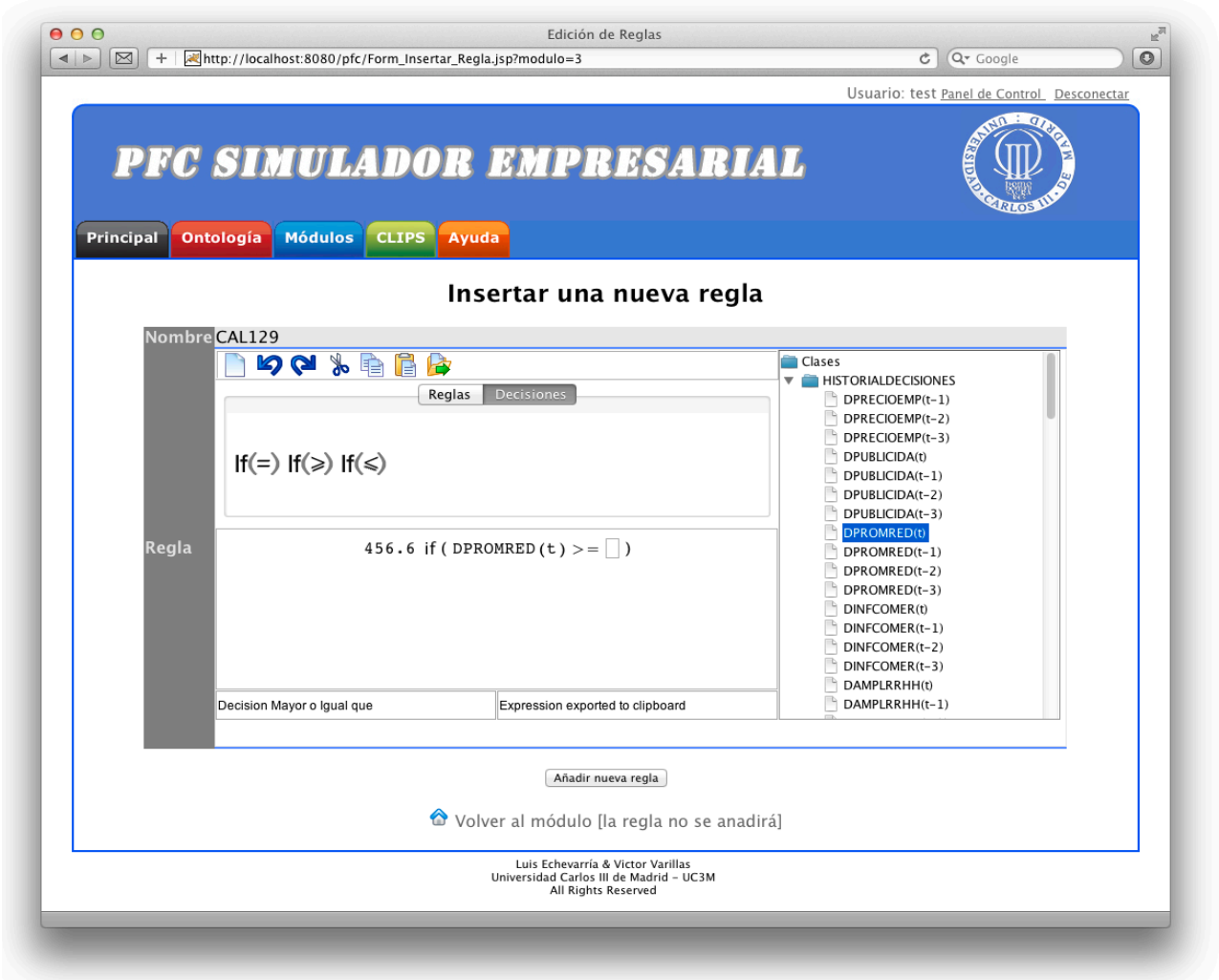


Figura 19: Inserción de Decisión

Otras acciones que aparecen en el editor están justo debajo del nombre del cálculo y son estas:

Estas siete acciones permiten (de izquierda a derecha):

- Borrar lo escrito y crear el cálculo desde cero.
- Deshacer
- Repetir
- Cortar (el trozo del cálculo seleccionado)
- Copiar (el trozo del cálculo seleccionado)
- Pegar
- Copiar expresión al portapapeles (en formato Maxima)

Para seleccionar un trozo de un cálculo sólo hay que arrastrar el puntero desde el inicio donde se quiere seleccionar hasta el final.

$$\frac{DPROMRED(t)+DPROMRED(t-2)}{456}$$

Una vez pulsada la opción *Añadir nueva regla* o *Modificar la regla,* según si se está creando o editando, y si no hay ningún problema aparece la siguiente pantalla.

Figura 20: Pantalla de finalización correcta

Si el cálculo que se ha introducido es incorrecto, ya sea porque hay un símbolo que no existe, porque un atributo no aparece en la ontología tal como se ha escrito o porque hay un error de sintaxis se mostrará un mensaje de error al usuario indicando el motivo y, en los casos en los que proceda, la parte a partir de la que el cálculo empieza a tener fallos. Por ejemplo:

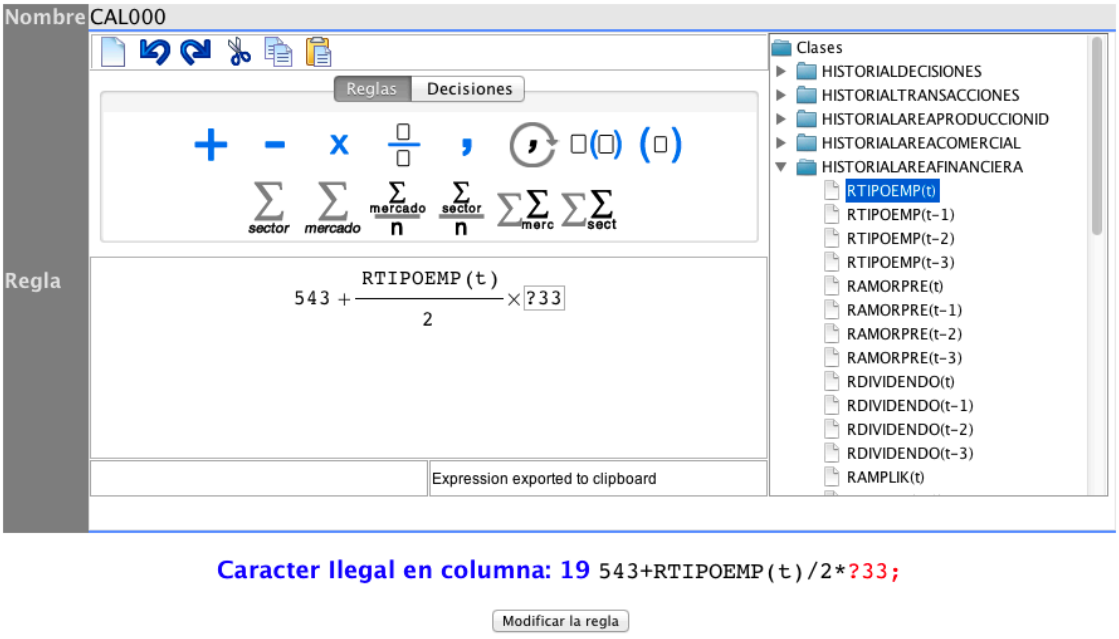


Figura 21: Pantalla de error sintáctico

En este caso aparece el símbolo "?" que es un carácter que no está dentro del léxico permitido.

Generación de los nuevos ficheros

Una vez que se tienen los cálculos creados dentro de sus correspondientes módulos sólo falta obtener los ficheros generadorPredicciones.clp y ontología-genérica.clp. Para ello hay que ir a la pestaña *CLIPS* donde se verá esto:

Figura 22: Pantalla de generación de ficheros

Tras seguir los pasos, finalmente aparecerán ambos archivos disponibles para descargar (si se pulsa en descargar fichero), como para ver en la misma ventana si se pulsa directamente sobre las imágenes de los disquetes.

Figura 23: Pantalla de obtención de ficheros

Ayuda

Bajo la pestaña de *Ayuda* se encuentra un pequeño manual de referencia rápida para usar la aplicación.

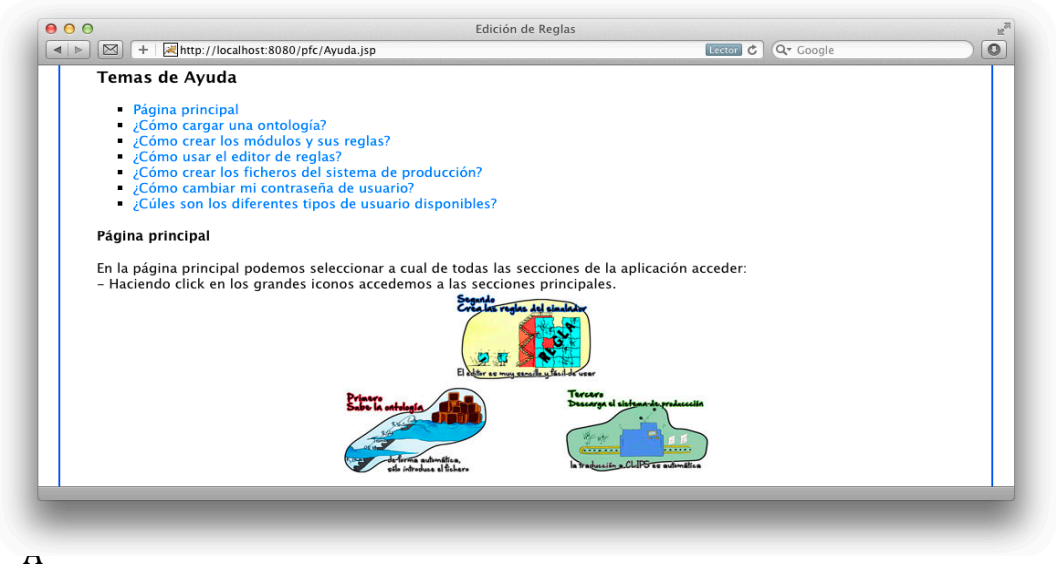


Figura 24: Ventana de ayuda

Arriba a la derecha aparece la información del usuario con el que se está trabajando. Si se ha acabado de usar la aplicación se pulsará *Desconectar* para volver a la página de autenticación original.

En el panel de control es posible cambiar la contraseña de acceso para el usuario con el que se esté autenticado en ese momento.

Figura 25: Pantalla de gestión de cuenta

9.2 MANUAL DE INSTALACIÓN

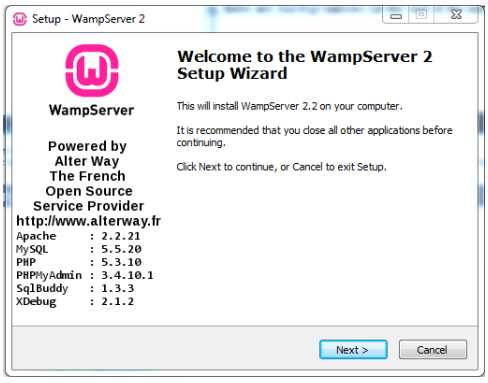
A continuación se describen todos los pasos necesarios para instalar la aplicación en el servidor. La instalación puede realizarse en cualquier plataforma que soporte **Apache Tomcat** y una base de datos **SQL**. A modo de ejemplo a continuación se describe como instalar la aplicación en Windows, usando la infraestructura **WAMP** (Apache, MySQL y PHP en Windows) con el fin de que el administrador tenga una interfaz gráfica (phpMyAdmin, incluido en WAMP) con la que controlar la base de datos.

Por motivos técnicos actualmente la configuración de la base de datos debe ser fija. La configuración de la base de datos debe ser localhost\pfc usuario:root password:root

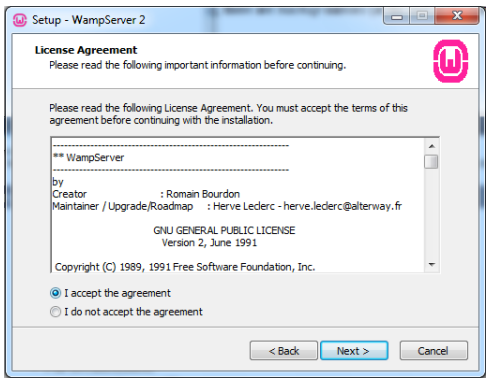
Instalación WAMP

Descarga el fichero **wampserver2.2d-x32.exe** a una carpeta local

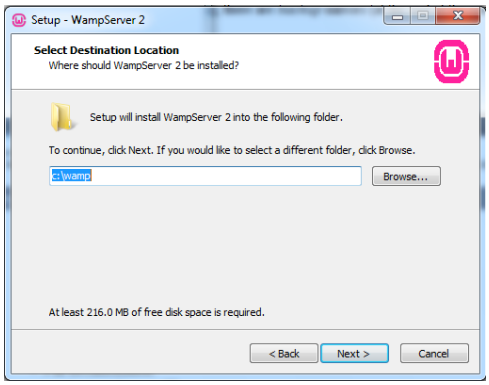
Ejecuta **wampserver2.2d-x32.exe**



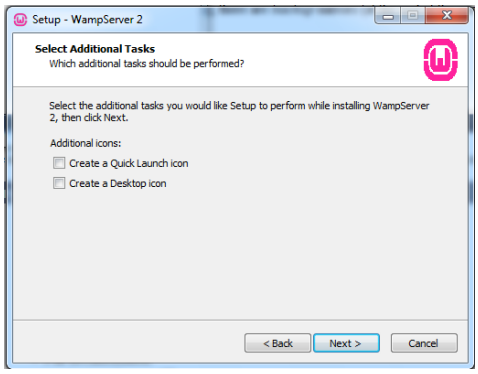
En la pantalla de bienvenida pulsa **Next>**



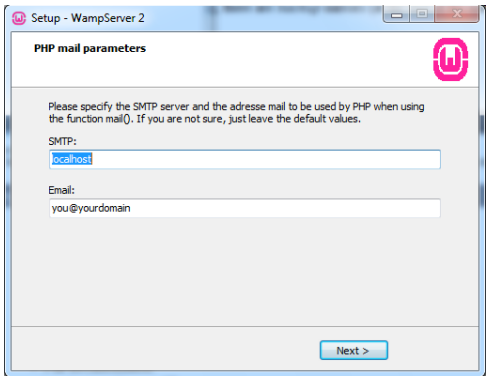
Selecciona la opción **"I accep the agreement"** y pulsa **Next>**



Por defecto WAMP se instala en C:\wamp
Si deseas instalarlo en otro lugar pulsa **Browse** y selecciona la carpeta donde se instalará y pulsa **OK**
Cuando la ruta de instalación sea la correcta pulsa **Next>**



Pulsa **Next>**



Pulsa **Next>**

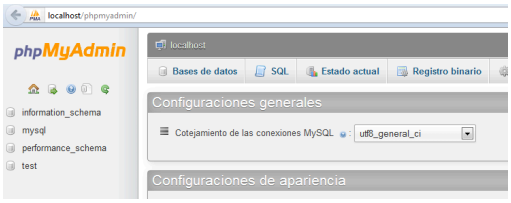
Selecciona **"Launch WampServer 2 now"** y pulsa **Finish.**

Creación de las tablas de la base de datos

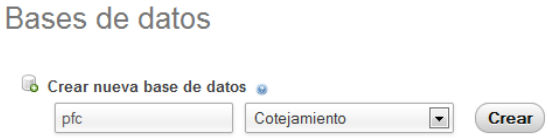
Para crear las tablas de la base de datos son necesarios 2 simples pasos:

- Crea una base de datos en tu base de datos MySQL
- Ejecuta el script de creación incluido en la documentación del proyecto llamado "creación_tablas_bbdd.sql"

A modo de ejemplo, se explica como crear las tablas con la base de datos anteriormente instalada.

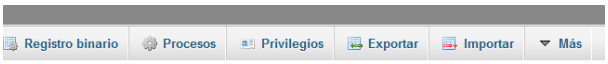


Abre la página de phpMyAdmin de la base de datos anteriormente instalada. En el menú superior haz click en **Bases de datos.**

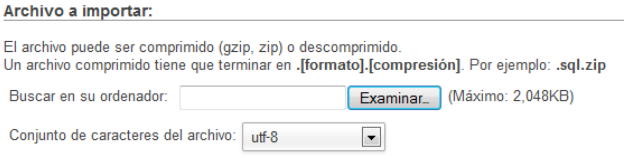


Introduce el nombre de la nueva base de datos y pulsa **crear**

En el menú de la izquierda selecciona la nueva base de datos creada.



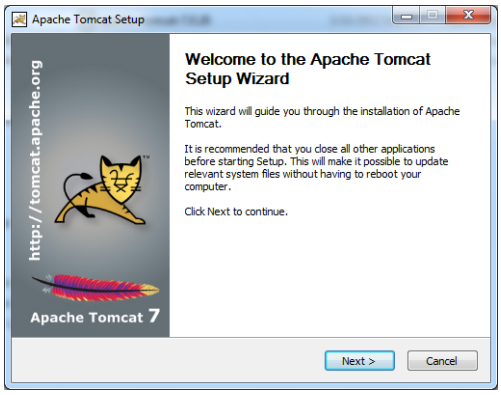
En la parte superior del menú, **haz click en Importar**



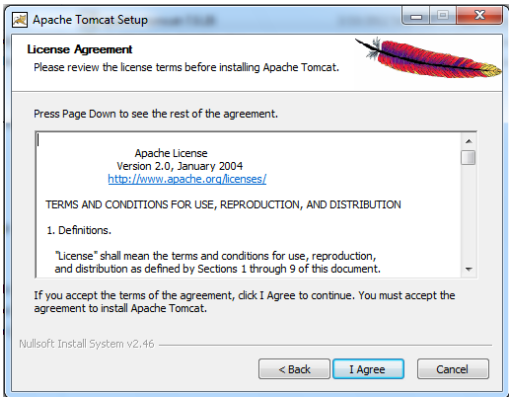
Haz click en Examinar y selecciona el archivo con el script de creación de la base de datos y pulsa **Continuar**

Instalación Apache Tomcat

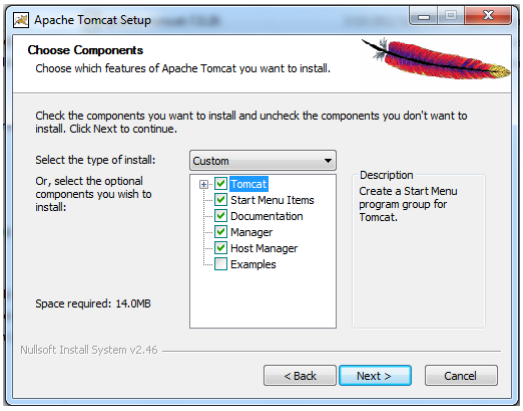
Antes de comenzar la instalación Java SE 6.0 o posterior debe haber sido instalado en el servidor. Se puede descargar desde la [web de Java]



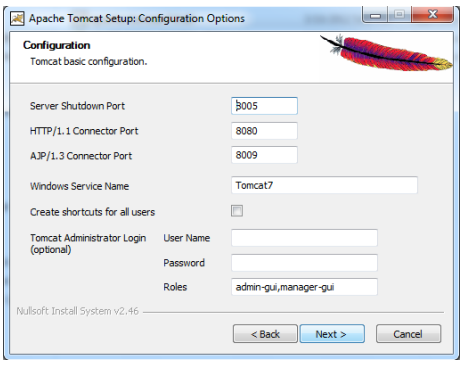
Ejecuta el archivo de instalación.
Haz click en **Next>**



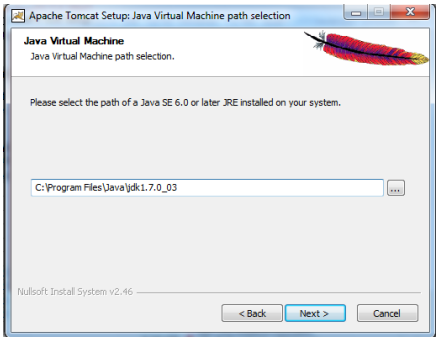
Haz click en **"I Agree"**



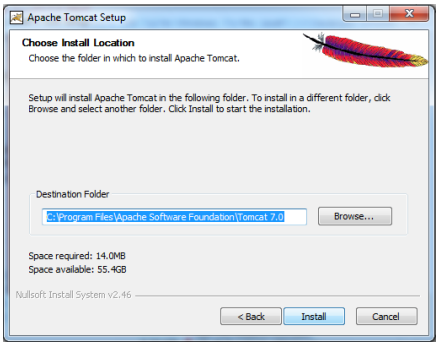
Haz click en **Next>**



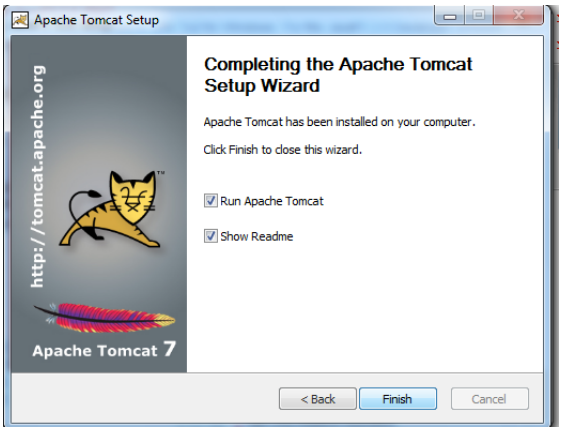
Haz click en **Next>**



Selecciona el directorio donde está instalado Java y haz click en **Next>**



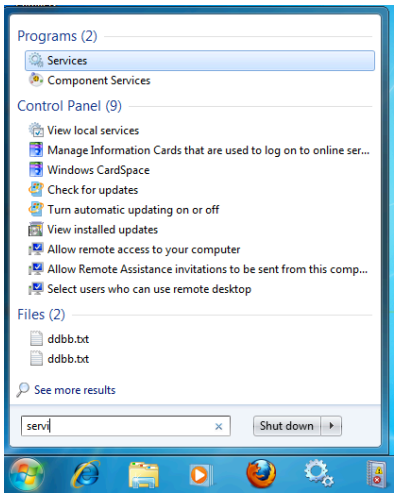
Selecciona la ruta de instalación y haz click en **Next>**



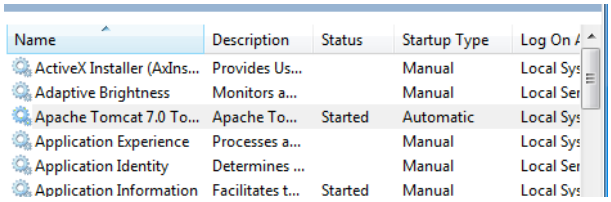
Haz click en **Finish**

Configuración Apache Tomcat

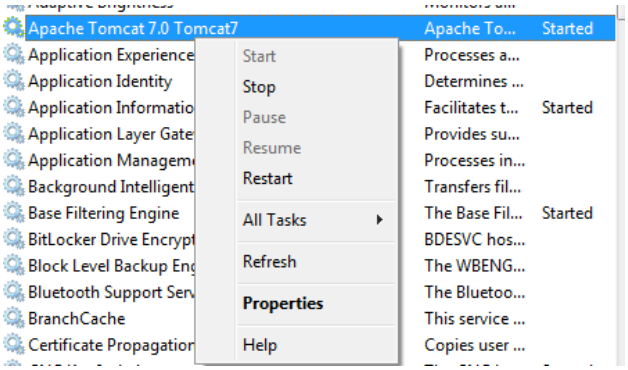
Iniciar/Parar Servicios



Haz click en el boto de Inicio y escribe "services" .
Haz click en la **consola de servicios**



Busa el servicio "Apache Tomcat 7.0 Tomcat 7" y haz click con el botón derecho encima del nombre.



Haz click en **Stop/Start**

Configurar usuarios e instalación de la aplicación

Copia la carpeta pfc de **binaries/tomcat/pfc** a **Tomcat\webapps**
Una vez copiada la carpeta, para los servicios.

En el archivo /Tomcat/conf/server.xml introduce, entre "<Engine name="Catalina" defaultHost="localhost">", el siguiente código:

```
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
        driverName="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://BBDD:PORT/PFC?user=USER&amp;password=PASS"
userTable="users" userNameCol="user_name" userCredCol="user_pass"
userRoleTable="user_roles" roleNameCol="role_name"/>
```

Siendo:

BBDD: dirección de la base de datos. Ejemplo: localhost
PORT: el puerto de la base de datos MySQL. Ejemplo: 8889
USER: el usuario para acceder a la base de datos. Ejemplo: root
PASS: password del usuario de la bas de datos. Ejemplo: root

Nota: Siendo la carpeta Tomcat donde se ha instalado el servidor Apache Tomcat

De esta manera se consigue conectar la base de datos con Apache. Ahora sólo falta definir la base de datos como un recurso válido para la aplicación.

La aplicación posee un archivo de configuración propio donde se definen entre otros la URL de la aplicación. Aquí se define la conexión con la base de datos.

Para ello, edita el archivo Tomcat\conf\Catalina\localhost\pfc.xml

El fichero quedará de la siguiente manera:

```
<Context path="PATH_URL" docBase="PATH" debug="1" reloadable="true"
crossContext="true">


        <Logger className="org.apache.catalina.logger.FileLogger" prefix="PFC_log."
suffix=".txt" timestamp="true"/>


        <Resource name="jdbc/PFC" auth="Container" type="javax.sql.DataSource"
driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://BBDD:PORT/PFC"
username="USER" password="PASS" maxActive="100" />
```

Siendo:

PATH_URL: La dirección url de la aplicación. Ejemplo: "/pfc"
PATH: Donde se encuentra en local la carpeta con los archivos. Ejemplo: "/Users/luisecheva/ITIG/PFC/apache-tomcat-7.0.4/webapps/pfc"
BBDD: Dirección de la base de datos. Ejemplo: localhost
PORT: El puerto de la base de datos MySQL. Ejemplo: 8889
USER: El usuario para acceder a la base de datos. Ejemplo: root
PASS: Password del usuario de la base de datos. Ejemplo: root

Una vez realizados todos estos pasos, la aplicación ya está lista para ser usada. Simplemente hay que asegurarse de tener usuarios creados en la base de datos.

A modo de ejemplo se recuerdan los roles disponibles:

- **admin-gui**: Rol de la administración de tomcat.
- **manager-gui**: Rol de acceso al Manager de tomcat para gestionar las diferentes aplicaciones.
- **usuario**: Rol para la lectura/edición de las reglas, importar ontología y descarga de los ficheros.
- **admin**: Rol para administración, tiene los máximos permisos en la aplicación.
- **lectura**: Rol para la lectura de las reglas y descarga de los ficheros.

9.3 PRUEBAS DE VERIFICACIÓN DE REGLAS

En este anexo se encuentran todas las pruebas que se han realizado en el editor a la hora de introducir reglas. Si la regla es "traducible" a CLIPS por el editor el resultado debería ser satisfactorio y la regla se añadiría a la BBDD. En caso contrario debería aparecer un mensaje de error junto con una breve descripción.

PRUEBA 1	
Entrada	Regla vacía
Resultado esperado	Error de regla vacía
Resultado obtenido	ERROR: La regla introducida está vacía

PRUEBA 2	
Entrada	543 (Número natural cualquiera)
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 3	
Entrada	0.0 (Número decimal con punto)
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 4	
Entrada	123,0 (Número decimal con coma)
Resultado esperado	Error léxico
Resultado obtenido	Carácter Ilegal <,> en columna: 4

PRUEBA 5	
Entrada	123+123.1 (Suma de números decimales)
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 6	
Entrada	-(1.2)-((-(3))-2) (Operaciones con números decimales y negativos)
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 7	
Entrada	/12 (Operador sin un operando delante)
Resultado esperado	Error sintáctico
Resultado obtenido	Error en columna 1: Syntax error

PRUEBA 8	
Entrada	123?
Resultado esperado	Error léxico
Resultado obtenido	Carácter ilegal en columna: 4

PRUEBA 9	
Entrada	DPRECIOEMP(t)/DPRECIOEMP(t-1) (Suma de Atributos)
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 10	
Entrada	DPRECIOEMP(t)/DPRECIO(t-1) (División de Atributos erróneos)
Resultado esperado	El atributo introducido no existe
Resultado obtenido	ERROR: El Atributo DPRECIO(t-1) no existe en la Ontología

PRUEBA 11	
Entrada	$\sum_{\text{sector}} \text{REXCESO}(t)$ Sumatorio de Atributo con información histórica.
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 12	
Entrada	$\sum_{\text{mercado}} \text{PIPC}(t)$ Sumatorio de Atributo sin información histórica.
Resultado esperado	Mensaje de error. No se puede calcular un sumatorio de un atributo cuya clase no tiene información histórica
Resultado obtenido	ERROR: El atributo PIPC(t) introducido no puede ir en un sumatorio

PRUEBA 13	
Entrada	$\frac{\sum_{\text{mercado}} \text{CAL080}(t)}{n}$ Sumatorio de un cálculo realizado anteriormente.
Resultado esperado	Mensaje de error. No es posible calcular sumatorios de cálculos anteriores.
Resultado obtenido	ERROR: El atributo CAL080(t) introducido no puede ir en un sumatorio

PRUEBA 14	
Entrada	$\sum_{t-3}^{t} \sum_{\text{mercado}} 5.9$ Sumatorio de un número
Resultado esperado	Mensaje de error. No es posible calcular el sumatorio de un número. Sólo de atributos de la Ontología
Resultado obtenido	ERROR: El atributo 5.9 introducido no puede ir en un sumatorio

PRUEBA 15	
Entrada	$\frac{\sum \text{RAGREGA}(t)}{\text{sector}}{n}$ Sumatorio del atributo de todas las empresas del sector entre el número de empresas del sector
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 16	
Entrada	$\frac{\sum \text{RAMPLIK}(t)}{\text{mercado}}{-(n)}$ Sumatorio de un atributo dividido entre el número de empresas en negativo
Resultado esperado	Mensaje de error. El número de empresas no puede ser negativo. El "-" es un carácter no válido en este contexto
Resultado obtenido	Carácter Ilegal en columna: 31

PRUEBA 17	
Entrada	Redondear (7.0)
Resultado esperado	Mensaje de error. La regla Redondear debe utilizarse para atributos de la Ontología
Resultado obtenido	ERROR: La regla REDONDEAR debe llevar un Atributo de la Ontología

PRUEBA 18	
Entrada	Redondear(-(RTIPOEMP)(t-2))
Resultado esperado	Mensaje de error. Sólo es posible Redondear los atributos como aparecen en la Ontología.
Resultado obtenido	Error en columna 11 : Syntax error

PRUEBA 19	
Entrada	Redondear((RTIPOE(t-2))
Resultado esperado	Mensaje de error. El Atributo RTIPOE no está en la Ontología.
Resultado obtenido	ERROR: El atributo RTIPOE(t-2) no existe en la Ontología

PRUEBA 20	
Entrada	Redondear(RVTANETA(t-2))
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 21	
Entrada	-5 if (RPLANTAE(t) = RPLANTAE (t))
Resultado esperado	Correcto
Resultado obtenido	Se ha añadido a la Base de Datos

PRUEBA 22	
Entrada	RTIPOEMP(t) if (RAMORPRE(t) = RAMPLIK (t))
Resultado esperado	Mensaje de error. El nuevo valor de la decisión debe ser un número. No puede ser un atributo.
Resultado obtenido	ERROR: El nuevo valor debe ser un número

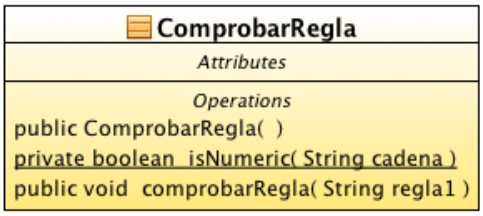
PRUEBA 23	
Entrada	7.0 if (RAORPRE(t) = RAMPLIK (t))
Resultado esperado	Mensaje de error. El atributo RAORPRE (t) no existe en la Ontología.
Resultado obtenido	ERROR: El atributo RAORPRE(t) no existe en la Ontología

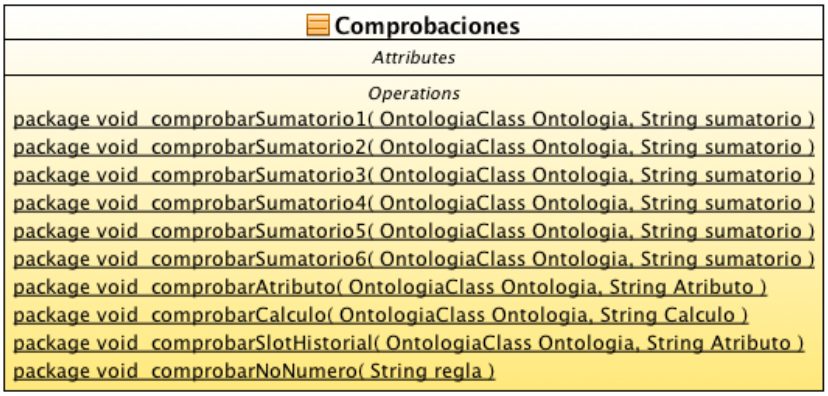
PRUEBA 24	
Entrada	6/6 if (RPLANTAE(t) = RPLANTAE (t))
Resultado esperado	Mensaje de error. Se ha de introducir un número. No una operación.
Resultado obtenido	Syntax error

9.4 MODELO DE CLASES

PAQUETE "GRAMMAR PACKAGE" (GRAMÁTICA DE VERIFICACIÓN)

Estas son las clases que integran el paquete dedicado a comprobar en el editor que un cálculo a grabar es correcto.





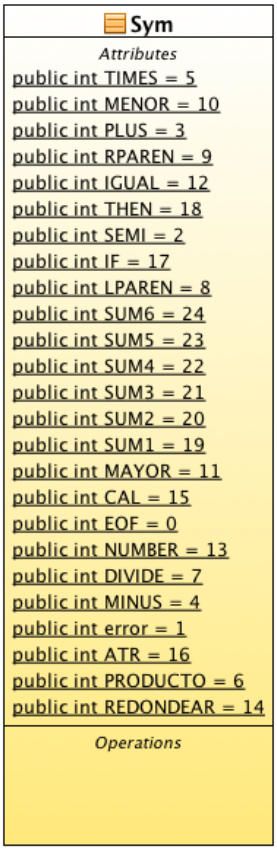
Lexer

Attributes

public int YYEOF = -1
private int ZZ_BUFFERSIZE = 16384
public int YYINITIAL = 0
private int ZZ_LEXSTATE[0..*] = {0, 0}
private String ZZ_CMAP_PACKED = "\11\0\1\3\1\2\1\0\1\3\1\1\1\22\0\1\3\7\0\1\7" + "\1\17\
private char ZZ_CMAP[0..*] = zzUnpackCMap(ZZ_CMAP_PACKED)
private int ZZ_ACTION[0..*] = zzUnpackAction()
private String ZZ_ACTION_PACKED_0 = "\1\0\1\1\2\2\1\1\3\1\1\1\4\1\5" + "\1\6\1\7\1\1\
private int ZZ_ROWMAP[0..*] = zzUnpackRowMap()
private String ZZ_ROWMAP_PACKED_0 = "\0\0\0\52\0\124\0\52\0\176\0\250\0\52\0\322" + "
private int ZZ_TRANS[0..*] = zzUnpackTrans()
private String ZZ_TRANS_PACKED_0 = "\1\2\1\3\2\4\1\5\2\6\1\7\1\2\2\6" + "\1\10\2\6\1\1]
private int ZZ_UNKNOWN_ERROR = 0
private int ZZ_NO_MATCH = 1
private int ZZ_PUSHBACK_2BIG = 2
private String ZZ_ERROR_MSG[0..*] = {"Unkown internal scanner error", "Error: could not match inpu
private int ZZ_ATTRIBUTE[0..*] = zzUnpackAttribute()
private String ZZ_ATTRIBUTE_PACKED_0 = "\1\0\1\11\1\1\1\11\2\1\1\11\2\1\3\11" + "\1\1\6
private Reader zzReader
private int zzState
private int zzLexicalState = YYINITIAL
private char zzBuffer[0..*] = new char[ZZ_BUFFERSIZE]
private int zzMarkedPos
private int zzCurrentPos
private int zzStartRead
private int zzEndRead
private int yyline
private int yychar
private int yycolumn
private boolean zzAtBOL = true
private boolean zzAtEOF
private boolean zzEOFDone

Operations

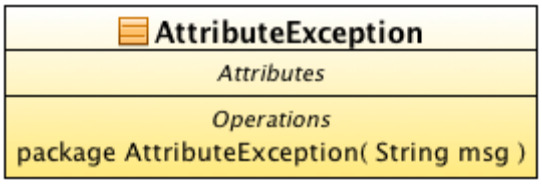
private int[0..*] zzUnpackAction( )
private int zzUnpackAction(String packed, int offset, int result[0..*])
private int[0..*] zzUnpackRowMap( )
private int zzUnpackRowMap(String packed, int offset, int result[0..*])
private int[0..*] zzUnpackTrans( )
private int zzUnpackTrans(String packed, int offset, int result[0..*])
private int[0..*] zzUnpackAttribute( )
private int zzUnpackAttribute(String packed, int offset, int result[0..*])
public Lexer(String inputStr)
private Symbol symbol(int type)
private Symbol symbol(int type, Object value)
package Lexer(Reader in)
package Lexer(InputStream in)
private char[0..*] zzUnpackCMap(String packed)
private boolean zzRefill( )
public void yyclose( )
public void yyreset(Reader reader)
public int yystate( )
public void yybegin(int newState)
public String yytext( )
public char yycharat(int pos)
public int yylength( )
private void zzScanError(int errorCode)
public void yypushback(int number)
private void zzDoEOF( )
public Symbol next_token( )

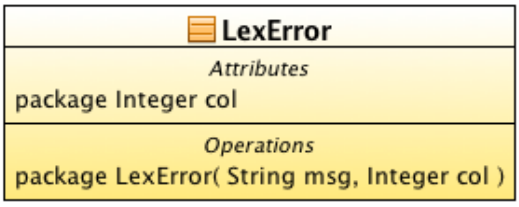


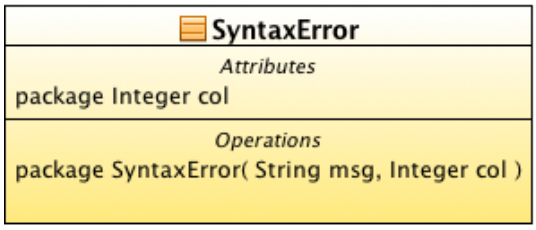
Las clases están ordenadas según van tomando un papel importante en el proceso de verificación. Primero se llama a la clase referenciada desde el editor donde se llama al método que comienza comprobando el cálculo. Se realizan una serie de comprobaciones preliminares que pueden derivar en un "***AttributeException***" en caso de que alguna no se cumpla.

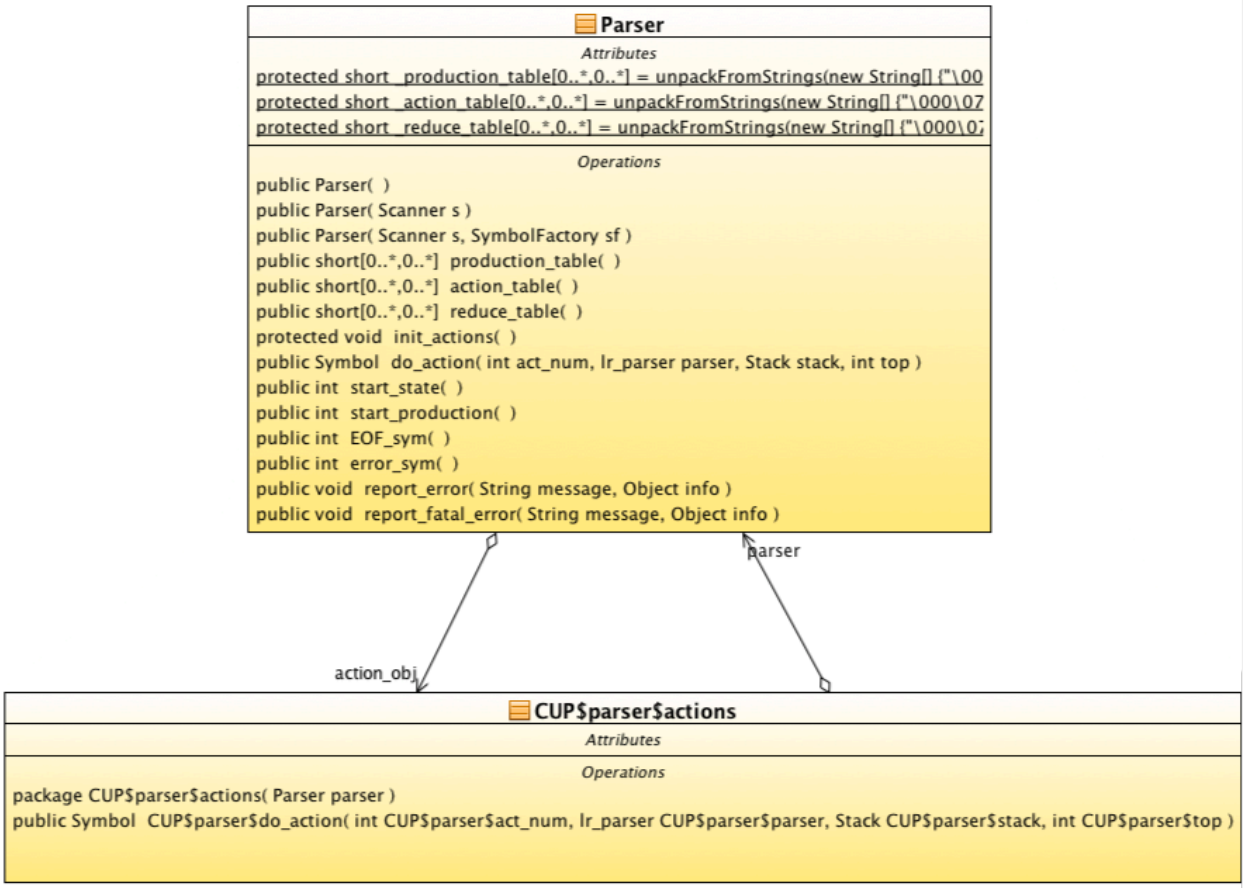
A continuación se realiza el análisis léxico en la clase "***Lexer***" que divide la entrada en los *tokens* identificados en la clase "***Sym***". Si el *token* no se encuentra se lanza un Error "***LexError***".

Después se realiza el análisis sintáctico dentro de la clase "*Parser*" que devolverá un "***SyntaxError***" en caso de que el cálculo no coincida con alguna de las producciones de la gramática.



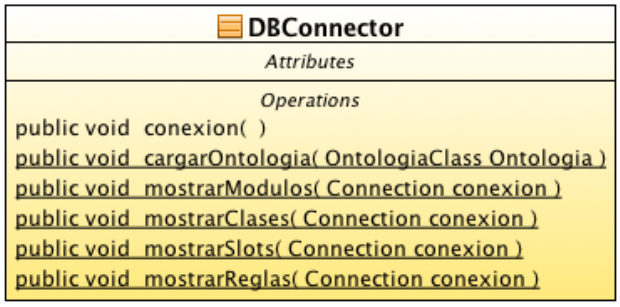




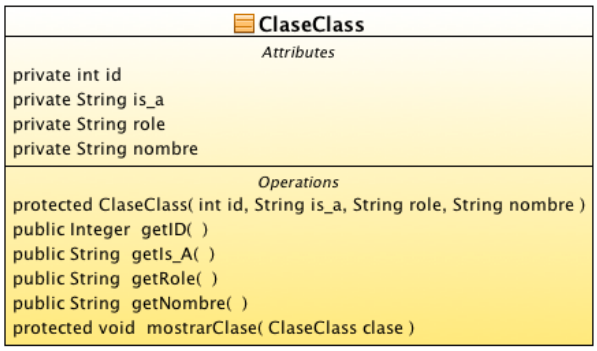

Parser
Attributes
protected short _production_table[0..*,0..*] = unpackFromStrings(new String[] {"\00
protected short _action_table[0..*,0..*] = unpackFromStrings(new String[] {"\000\07
protected short _reduce_table[0..*,0..*] = unpackFromStrings(new String[] {"\000\07
Operations
public Parser( )
public Parser(Scanner s)
public Parser(Scanner s, SymbolFactory sf)
public short[0..*,0..*] production_table( )
public short[0..*,0..*] action_table( )
public short[0..*,0..*] reduce_table( )
protected void init_actions( )
public Symbol do_action(int act_num, lr_parser parser, Stack stack, int top)
public int start_state( )
public int start_production( )
public int EOF_sym( )
public int error_sym( )
public void report_error(String message, Object info)
public void report_fatal_error(String message, Object info)
parser
action_obj
CUP$parser$actions
Attributes
Operations
package CUP$parser$actions(Parser parser)
public Symbol CUP$parser$do_action(int CUP$parser$act_num, lr_parser CUP$parser$parser, Stack CUP$parser$stack, int CUP$parser$top)

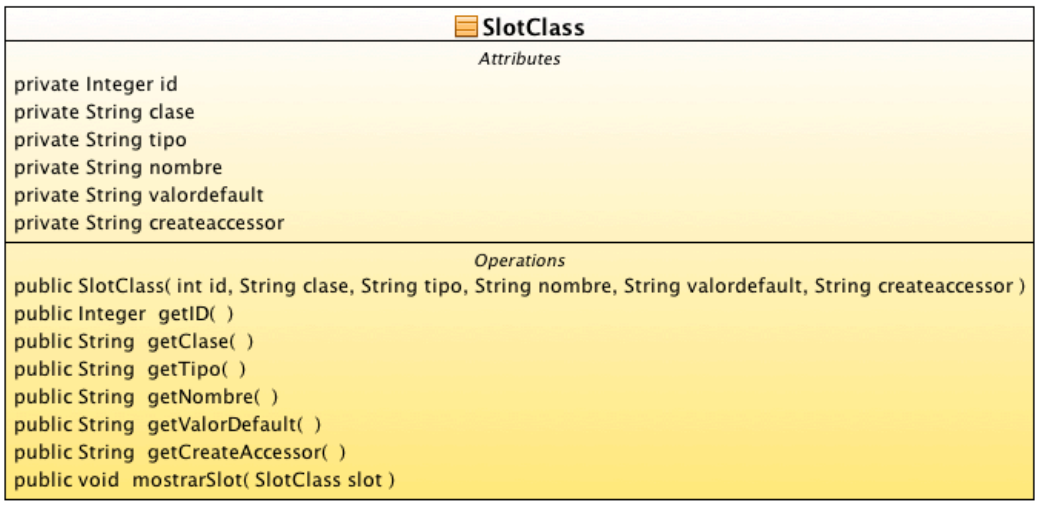
PAQUETE "ONTOLOGIA PACKAGE" (MODELO DE LA INFORMACIÓN EN LA BBDD)

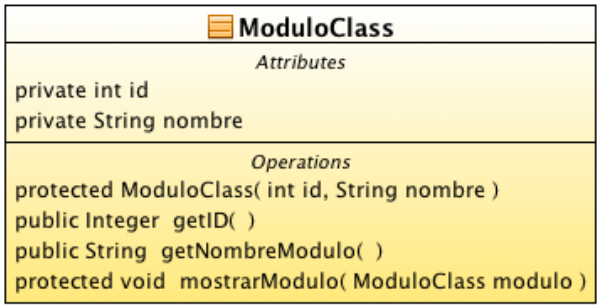
Este paquete se centra en **cargar y almacenar la información contenida en la BBDD** que hace de interfaz con el editor una vez que el usuario ha acabado de trabajar con este. Lo primero es una clase que se encargue de gestionar las conexiones con la BBDD.

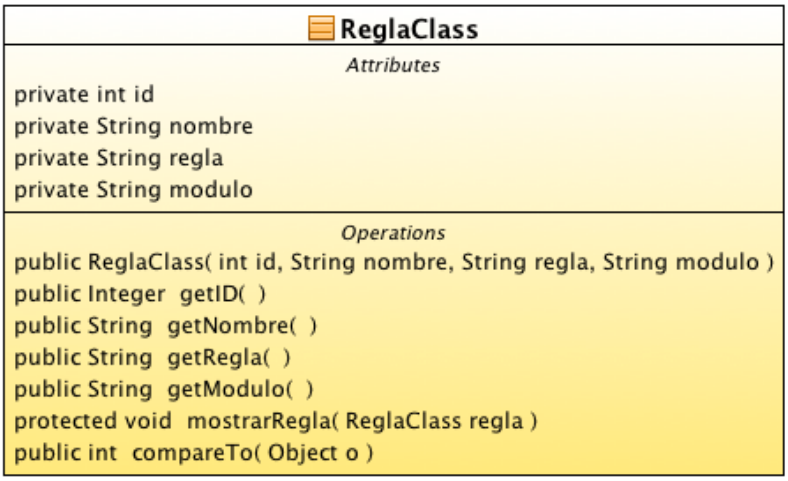


La información se almacena en la clase Ontología que guarda las listas con los cálculos dentro de sus módulos y las clases con sus Slots. Destacar que el diseño de este paquete tiene muchas formas de realizarse, como incluir dentro de la clase Ontología una lista con los módulos y otra con las clases y que la clase Módulos tuviese una lista de cálculos y la clase de Clase tuviese una lista con sus slots. Para una mayor sencillez a la hora de cargar la información de la BBDD se ha optado por este diseño.

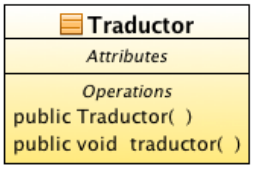

ClaseClass
Attributes
private int id
private String is_a
private String role
private String nombre
Operations
protected ClaseClass(int id, String is_a, String role, String nombre)
public Integer getID( )
public String getIs_A( )
public String getRole( )
public String getNombre( )
protected void mostrarClase(ClaseClass clase)

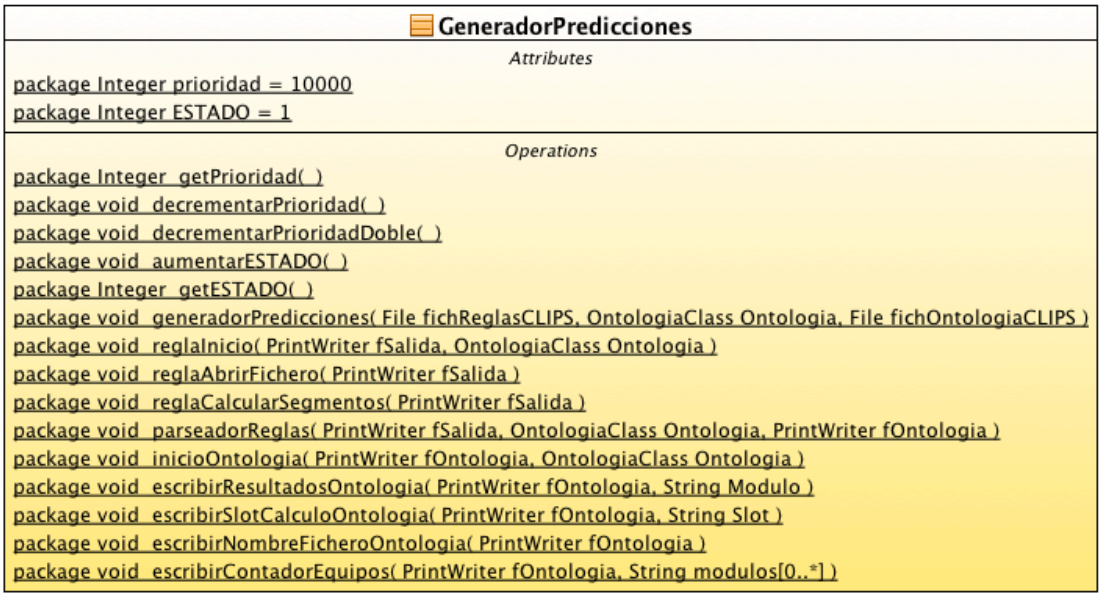

SlotClass
Attributes
private Integer id
private String clase
private String tipo
private String nombre
private String valordefault
private String createaccessor
Operations
public SlotClass(int id, String clase, String tipo, String nombre, String valordefault, String createaccessor)
public Integer getID( )
public String getClase( )
public String getTipo( )
public String getNombre( )
public String getValorDefault( )
public String getCreateAccessor( )
public void mostrarSlot(SlotClass slot)


ModuloClass
Attributes
private int id
private String nombre
Operations
protected ModuloClass(int id, String nombre)
public Integer getID( )
public String getNombreModulo( )
protected void mostrarModulo(ModuloClass modulo)

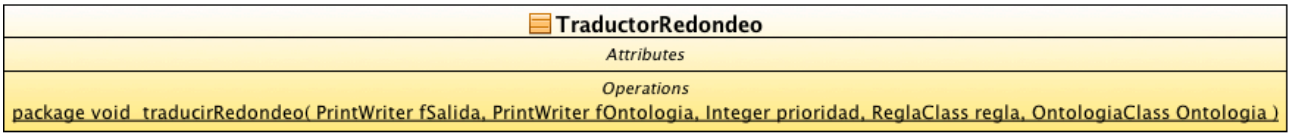

ReglaClass
Attributes
private int id
private String nombre
private String regla
private String modulo
Operations
public ReglaClass(int id, String nombre, String regla, String modulo)
public Integer getID( )
public String getNombre( )
public String getRegla( )
public String getModulo( )
protected void mostrarRegla(ReglaClass regla)
public int compareTo(Object o)

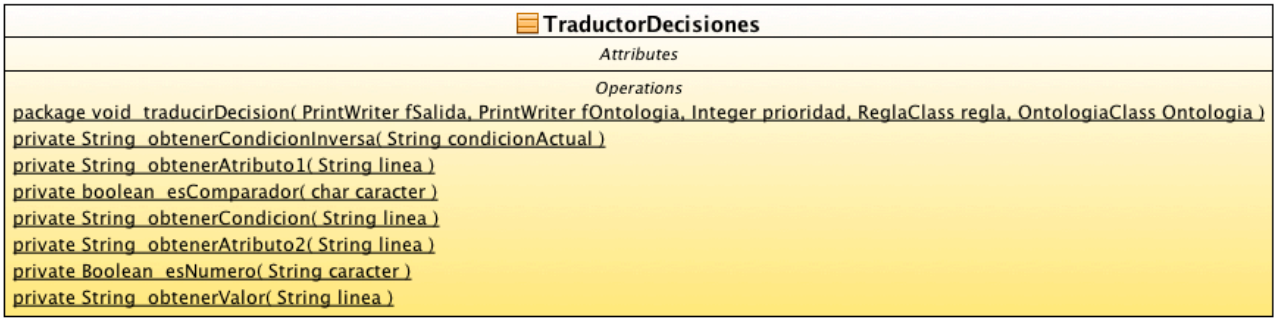
Este paquete es el encargado de ir tomando la información almacenada en el objeto Ontología e ir traduciendo los cálculos de la lista de cálculos. Primero se llama desde el editor a la clase *Traductor* que delegará en la clase ***GeneradorPredicciones***.



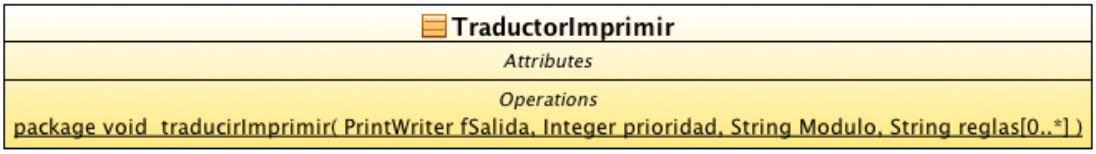


A ***GeneradorPredicciones*** le seguirá ***TraductorDecisiones*** o ***TraductorRedondeo*** cuando se trate de alguno de estos cálculos.

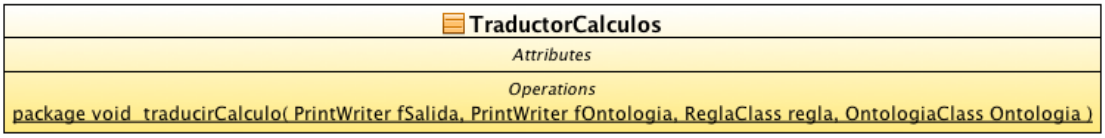




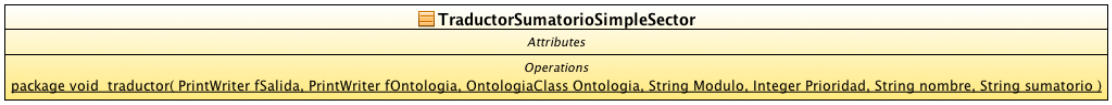
Si es el final de un módulo se crean las reglas en ***TraductorImprimir*** que muestran los resultados de este módulo.

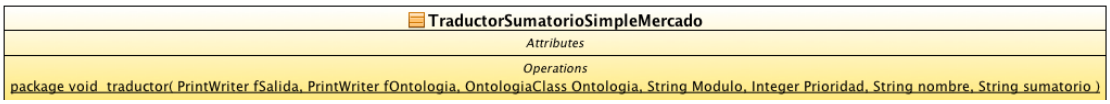


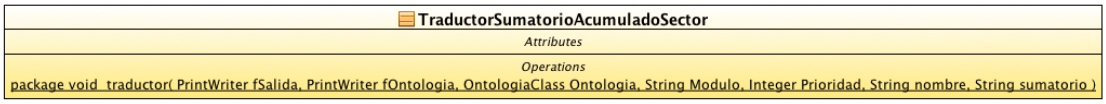
En el caso de que se trate un cálculo aritmético se llamará a ***TraductorCalculos***.

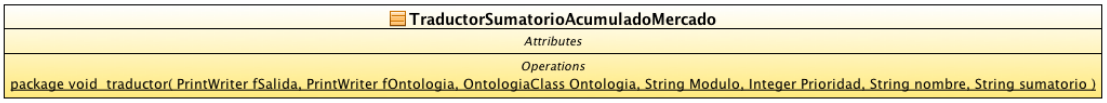


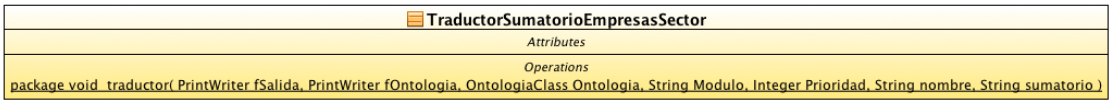
Si el cálculo es un sumatorio de los predefinidos en el editor se pasará a la clase correspondiente.

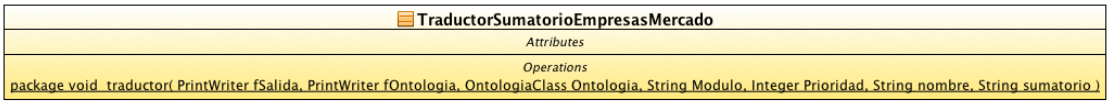






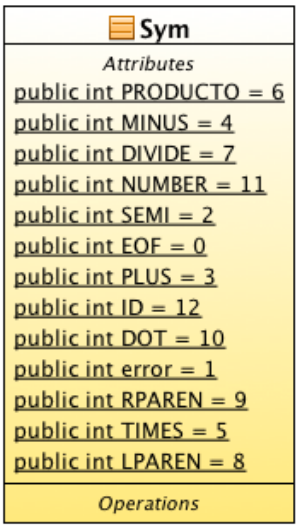


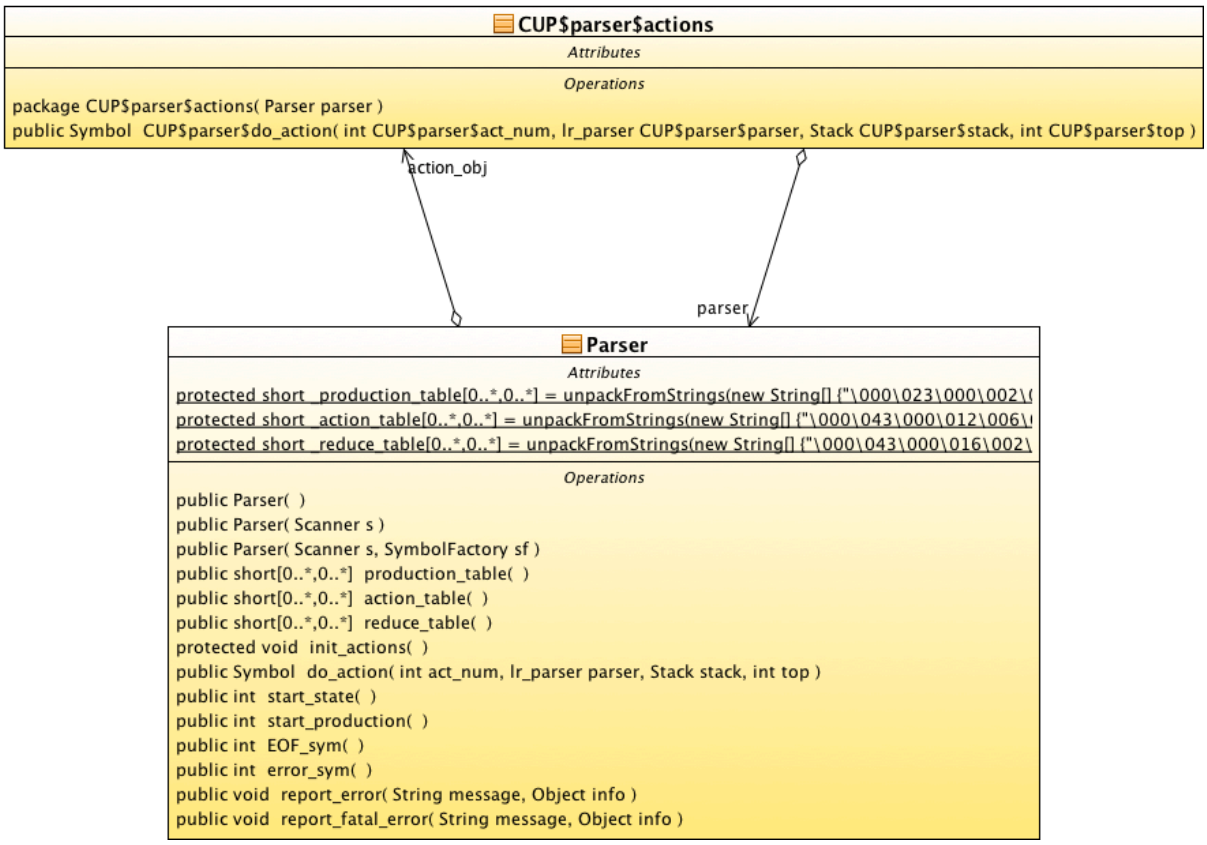




Después se deben tratar los cálculos para que queden reflejados en la regla CLIPS de manera prefija (operador operando operando). Para ello se realiza un análisis léxico para obtener los *token* (operandos y operadores).

Se obtienen los *token* identificados en la clase ***Sym***. Y por último se realiza el análisis sintáctico para que al encontrar un cálculo asociado a una producción de la gramática se lance la acción asociada a esa producción la cual devolverá el mismo cálculo en la notación prefija que es la usada en CLIPS.





9.5 DOCUMENTACIÓN JAVADOC DE LA IMPLEMENTACIÓN

Ontologia Package

Constructor Detail

DBConnector

```
public DBConnector()
```

Method Detail

conexion

```
public void conexion()
```

Método que conecta con la BBDD

---

cargarOntologia

```
public static void cargarOntologia(OntologiaClass Ontologia)
```

Este método conecta con la BBDD y carga en el objeto Ontologia los datos existentes de la BBDD

Parameters:
`Ontologia` - Objeto Ontologia que vamos a rellenar con la información cargada de la BBDD

---

mostrarModulos

```
public static void mostrarModulos(java.sql.Connection conexion)
```

Método que muestra los Módulos de la BBDD

Parameters:
`conexion` - Conexión a la BBDD

---

mostrarClases

```
public static void mostrarClases(java.sql.Connection conexion)
```

Método que muestra las Clases de la BBDD

Parameters:
`conexion` - Conexión con la BBDD

---

mostrarSlots

```
public static void mostrarSlots(java.sql.Connection conexion)
```

Método que muestra los Slots de la BBDD

Parameters:
`conexion` - Conexión con la BBDD

---

mostrarReglas

```
public static void mostrarReglas(java.sql.Connection conexion)
```

Método que muestra las Reglas de la BBDD

Constructor Detail

OntologiaClass

```
public OntologiaClass()
```

Constructor de la clase Ontologia

Method Detail

obtenerClaseSlot

```
public java.lang.String obtenerClaseSlot(java.lang.String Slot)
```

Método que devuelve la clase a la que pertenece el Slot que pasamos por parámetro

Parameters:
`Slot` - Slot del que queremos obtener su clase
Returns:
String - Devuelve la clase a la que pertenece el Slot

---

encontrarNombreRegla

```
public boolean encontrarNombreRegla(java.lang.String Nombre)
```

Método para comprobar que existe una regla con el nombre que buscamos

Parameters:
`Nombre` - Nombre de la regla a buscar
Returns:
Boolean - Booleano que nos indica si existe o no una regla con ese nombre

---

existenReglasEnModulo

```
public boolean existenReglasEnModulo(ModuloClass modulo)
```

Comprueba si hay reglas para el módulo pasado por parámetro

Parameters:
`modulo` - Módulo del que queremos ver si hay reglas
Returns:
Boolean - Valor verdadero si hay reglas, falso en caso contrario

---

obtenerPeriodoAtributo

```
public java.lang.String obtenerPeriodoAtributo(java.lang.String Atributo)
```

Método que devuelve el período de un determinado atributo

Parameters:
`Atributo` - El Slot del que queremos averiguar su período
Returns:
String - El período al que pertenece el Slot

obtenerSlot

```
public java.lang.String obtenerSlot(java.lang.String Atributo)
```

Método que devuelve el nombre del Slot sin el período

Parameters:
`Atributo` - Es el slot del que queremos extraer su nombre sin el período
Returns:
String - Es el nombre del slot sin el período

---

getModulo

```
public java.lang.String getModulo(int ID)
```

Método que devuelve el nombre del módulo pasándole el ID de módulo

Parameters:
`ID` - Es el identificador de módulo
Returns:
String - Devuelve el nombre del módulo

---

obtenerModulo

```
public java.lang.String obtenerModulo(java.lang.String mod)
```

Método que devuelve el nombre completo de un módulo pasándole su nombre corto

Parameters:
`mod` - Tres primeras letras del módulo que deseamos buscar
Returns:
String - Nombre completo del módulo

---

imprimirRestriccionPeriodo

```
public void imprimirRestriccionPeriodo(java.io.PrintWriter fSalida,
                                       java.lang.String Periodo)
```

Dado el período que se recibe se imprime en el fichero de salida la restricción que corresponda

Parameters:
`fSalida` - Fichero en el que se están escribiendo las reglas
`Periodo` - Período sobre el que se quiere escribir la restricción

---

anadirClase

```
public void anadirClase(ClaseClass Clase)
```

Añade una clase a la lista de clases de la ontología

Parameters:
`Clase` - Clase a añadir

---

anadirSlot

```
public void anadirSlot(SlotClass Slot)
```

Añade un Slot a la lista de Slots de la ontología

Parameters:
`Slot` - Slot a añadir

anadirModulo

```
public void anadirModulo(ModuloClass Modulo)
```

Añade un módulo a la lista de módulos de la ontología

Parameters:
`Modulo` - Módulo a añadir

---

anadirRegla

```
public void anadirRegla(ReglaClass Regla)
```

Añade una regla a la lista de reglas de la ontología

Parameters:
`Regla` - Regla a añadir

---

getListaClases

```
public java.util.ArrayList getListaClases()
```

Devuelve la lista de Clases de la ontología

Returns:
ArrayList - ListaClases La lista de clases

---

getListaSlots

```
public java.util.ArrayList getListaSlots()
```

Devuelve la lista de Slots de la ontología

Returns:
ArrayList - ListaSlots La lista de Slots

---

getListaModulos

```
public java.util.ArrayList getListaModulos()
```

Devuelve la lista de Módulos de la ontología

Returns:
ArrayList - Lista de Módulos

---

getListaReglas

```
public java.util.ArrayList getListaReglas()
```

Devuelve la lista de Reglas de la ontología

Returns:
ArrayList - La lista de Reglas

mostrarClases

```
public void mostrarClases(java.util.ArrayList listaClases)
```

Muestra la lista de Clases cargadas en la ontología

Parameters:
`listaClases` - Lista de Clases

---

mostrarSlots

```
public void mostrarSlots(java.util.ArrayList listaSlots)
```

Muestra la lista de Slots cargadas en la ontología

Parameters:
`listaSlots` - Lista de Slots

---

mostrarModulos

```
public void mostrarModulos(java.util.ArrayList listaModulos)
```

Muestra la lista de Módulos cargados en la ontología

Parameters:
`listaModulos` - Lista de Módulos

---

mostrarReglas

```
public void mostrarReglas(java.util.ArrayList listaReglas)
```

Muestra la lista de reglas cargadas en la ontología

Parameters:
`listaReglas` - Lista de reglas

---

escribirSlot

```
public void escribirSlot(java.io.PrintWriter fSalida,
                         OntologiaClass Ontologia,
                         java.lang.String Atributo)
```

Método que escribe el atributo que se pasa por parámetro en el fichero de salida junto c
restricciones que ayudan a encontrarlo en la Ontología

Parameters:
`fSalida` - Fichero de salida en el que escribiremos
`Ontologia` - Ontología genérica cargada de la BBDD
`Atributo` - Atributo que se quiere escribir en la BBDD

Constructor Detail

ClaseClass

```
protected ClaseClass(int id,
                     java.lang.String is_a,
                     java.lang.String role,
                     java.lang.String nombre)
```

Constructor de la clase Clase

Parameters:
- `id` - Identificador de clase
- `is_a` - Tipo de la clase
- `role` - Rol de la clase
- `nombre` - Nombre de la clase

Method Detail

getID

```
public java.lang.Integer getID()
```

Devuelve el identificador de la clase

Returns:
Integer - Identificador de la clase

---

getIs_A

```
public java.lang.String getIs_A()
```

Devuelve el tipo de clase

Returns:
String - Tipo de la clase

---

getRole

```
public java.lang.String getRole()
```

Devuelve el rol de la clase

Returns:
String - Rol de la clase

---

getNombre

```
public java.lang.String getNombre()
```

Devuelve el nombre de la clase

Returns:
String - Nombre de la clase

---

mostrarClase

```
protected void mostrarClase(ClaseClass clase)
```

Muestra información de la clase

Parameters:
- `clase` - Clase cuyos datos queremos mostrar

Constructor Detail

SlotClass

```
public SlotClass(int id,
                 java.lang.String clase,
                 java.lang.String tipo,
                 java.lang.String nombre,
                 java.lang.String valordefault,
                 java.lang.String createaccessor)
```

Constructor del Slot

Parameters:

`id` - Identificador del Slot
`clase` - Clase del Slot
`tipo` - Tipo del Slot
`nombre` - Nombre del Slot
`valordefault` - Valor por defecto del Slot
`createaccessor` - Modificador de acceso del Slot

Method Detail

getID

```
public java.lang.Integer getID()
```

Devuelve el ID del Slot

Returns:

String - Identificador del Slot

---

getClase

```
public java.lang.String getClase()
```

Devuelve la clase del Slot

Returns:

String - Clase del Slot

---

getTipo

```
public java.lang.String getTipo()
```

Devuelve el tipo del Slot

Returns:

String - Tipo del Slot

getNombre

```
public java.lang.String getNombre()
```

Devuelve el nombre del Slot

Returns:
String - Nombre del Slot

---

getValorDefault

```
public java.lang.String getValorDefault()
```

Devuelve el valor por defecto del Slot

Returns:
String - Valor por defecto del Slot

---

getCreateAccessor

```
public java.lang.String getCreateAccessor()
```

Devuelve el modificador de acceso del Slot

Returns:
String - Modificador de acceso del Slot

---

mostrarSlot

```
public void mostrarSlot(SlotClass slot)
```

Muestra la información del Slot

Parameters:
`slot` - Slot del que queremos mostrar la información

Constructor Detail

ModuloClass

```
protected ModuloClass(int id,
                      java.lang.String nombre)
```

Constructor de la clase Módulo

Parameters:
`id` - Identificador del módulo
`nombre` - Nombre del módulo

Method Detail

getID

```
public java.lang.Integer getID()
```

Devuelve el ID del módulo

Returns:
Integer - Identificador del módulo

---

getNombreModulo

```
public java.lang.String getNombreModulo()
```

Devuelve el nombre del módulo

Returns:
String - Nombre del módulo

---

mostrarModulo

```
protected void mostrarModulo(ModuloClass modulo)
```

Muestra la información del módulo

Parameters:
`modulo` - El módulo del que se desea mostrar la información

Constructor Detail

ReglaClass

```
public ReglaClass(int id,
                  java.lang.String nombre,
                  java.lang.String regla,
                  java.lang.String modulo)
```

Constructor de la clase Regla

Parameters:

`id` - Identificador de la regla
`nombre` - Nombre de la regla
`regla` - Contenido de la regla
`modulo` - Módulo al que pertenece la regla

Method Detail

getID

```
public java.lang.Integer getID()
```

Devuelve el identificador de la regla

Returns:

Integer - Identificador de la regla

---

getNombre

```
public java.lang.String getNombre()
```

Devuelve el nombre de la regla

Returns:

String - Nombre Nombre de la regla

---

getRegla

```
public java.lang.String getRegla()
```

Devuelve el contenido de la regla

Returns:

String - Regla Regla escrita en lenguaje Maxima

---

getModulo

```
public java.lang.String getModulo()
```

Devuelve el módulo al que pertenece la regla

Returns:
String - Modulo Módulo de la regla

---

mostrarRegla

```
protected void mostrarRegla(ReglaClass regla)
```

Muestra la regla que se pasa por parámetro

Parameters:
`regla` - Regla que se quiere mostrar

---

compareTo

```
public int compareTo(java.lang.Object o)
```

Método para establecer un orden en las reglas según el módulo al que pertenecen

Specified by:
`compareTo` in interface `java.lang.Comparable`

Parameters:
`o` - Objeto a comparar
Returns:
Integer - compareToIgnoreCase

Grammar Package

Constructor Detail

ComprobarRegla

```
public ComprobarRegla()
```

Constructor vacío usado para crear el objeto en el editor

Method Detail

comprobarRegla

```
public void comprobarRegla(java.lang.String regla1)
                    throws java.lang.Exception
```

Método llamado desde el traductor cada vez que se intenta grabar un cálculo para verificar si es correcto

Parameters:
`regla1` - Se recibe la regla a comprobar en lenguaje Maxima
Throws:
`java.lang.Exception` - Excepción cuyo mensaje indica el contenido del error

Constructor Detail

AttributeException

```
public AttributeException(java.lang.String msg)
```

Método que lanza una excepción propia donde indicamos el mensaje de error con el atributo

Parameters:
`msg` - Mensaje que se pasará al método de la clase padre para lanzar la excepción

Constructor Detail

LexError

```
public LexError(java.lang.String msg,
                java.lang.Integer col)
```

Método que lanza un Error léxico indicando la columna

Parameters:
`msg` - Mensaje que se pasará al método de la clase padre para lanzar la excepción
`col` - Número que indica la columna donde se ha producido el error léxico

Constructor Detail

SyntaxError

```
public SyntaxError(java.lang.String msg,
                   java.lang.Integer col)
```

Este método lanza un Error de tipo sintáctico indicando la columna donde se encuentra el error

Parameters:

`msg` - Mensaje que se pasará al método de la clase padre para lanzar la excepción
`col` - Número que indica la columna donde se encuentra el error sintáctico

Constructor Detail

Comprobaciones

```
public Comprobaciones()
```

Method Detail

comprobarSumatorio1

```
public static void comprobarSumatorio1(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que la clase del atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:

`Ontología` - Ontología cargada de la BBDD
`sumatorio` - Cálculo en lenguaje Maxima con el sumatorio

Throws:

`AttributeException` - Excepción que indica que el atributo es incorrecto

---

comprobarSumatorio2

```
public static void comprobarSumatorio2(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que el atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:

`Ontologia` - Ontología cargada de la BBDD
`sumatorio` - Cálculo en lenguaje Maxima con el sumatorio

Throws:

`AttributeException` - Excepción que indica que el atributo es incorrecto

---

comprobarSumatorio3

```
public static void comprobarSumatorio3(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que el atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`sumatorio` - Cálculo en lenguaje Maxima con el sumatorio
Throws:
`AttributeException` - Excepción que indica que el atributo es incorrecto

---

comprobarSumatorio4

```
public static void comprobarSumatorio4(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que el atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`sumatorio` - Cálculo en lenguaje Maxima con el sumatorio
Throws:
`AttributeException` - Excepción que indica que el atributo es incorrecto

---

comprobarSumatorio5

```
public static void comprobarSumatorio5(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que el atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`sumatorio` - Cálculo en lenguaje Maxima con el sumatorio
Throws:
`AttributeException` - Excepción que indica que el atributo es incorrecto

---

comprobarSumatorio6

```
public static void comprobarSumatorio6(OntologiaClass Ontologia,
                                       java.lang.String sumatorio)
                                throws AttributeException
```

Método que comprueba que el atributo del sumatorio tiene IDEMPRESA y PERIODO

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`sumatorio` - Regla en Maxima con el sumatorio
Throws:
`AttributeException` - Excepción que indica que el atributo es incorrecto

comprobarAtributo

```
public static void comprobarAtributo(OntologiaClass Ontologia,
                                      java.lang.String Atributo)
                               throws AttributeException
```

Método que comprueba que el atributo se encuentra en la Ontología

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`Atributo` - Atributo que vamos a buscar en la Ontología
Throws:
`AttributeException` - Error que indica que el atributo no está en la Ontolo

---

comprobarCalculo

```
public static void comprobarCalculo(OntologiaClass Ontologia,
                                    java.lang.String Calculo)
                             throws AttributeException
```

Método que comprueba que el cálculo está en la Ontología

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`Calculo` - Slot de un cálculo anterior que queremos buscar en la Ontología
Throws:
`AttributeException` - Error que indica que el cálculo anterior no existe

---

comprobarSlotHistorial

```
public static void comprobarSlotHistorial(OntologiaClass Ontologia,
                                          java.lang.String Atributo)
                                   throws AttributeException
```

Método para comprobar que el Slot forma parte de un Historial

Parameters:
`Ontologia` - Ontología cargada de la BBDD
`Atributo` - Atributo que deseamos buscar
Throws:
`AttributeException` - Mensaje de error indicando que el Slot no forma pa
de una clase HISTORIAL

---

comprobarNoNumero

```
public static void comprobarNoNumero(java.lang.String regla)
                               throws AttributeException
```

Éste método sirve para comprobar que el cálculo REDONDEAR lleva un atributo Ontología, no un número

Parameters:
`regla` - Cálculo en lenguaje Maxima que contien un redondeo
Throws:
`AttributeException` - Mensaje de error que indica que la regla REDONDEAR no lleva un atributo de la Ontología

Translator Package

Constructor Detail

Traductor

```
public Traductor()
```

Constructor vacío

Method Detail

traductor

```
public void traductor()
               throws java.lang.Exception
```

Método traductor. Es llamado desde el editor y da inicio a toda la traducción a reglas CLIPS

Throws:

`java.lang.Exception` - Excepción que contiene el mensaje de error si este llega a producirse

Field Detail

prioridad

```
public static java.lang.Integer prioridad
```

Variable que controlará la prioridad de las reglas. Valor Máximo: 10000 Valor mínimo: -10000

ESTADO

```
public static java.lang.Integer ESTADO
```

Variable que controla el flujo de ejecución

Constructor Detail

GeneradorPredicciones

```
public GeneradorPredicciones()
```

Method Detail

getPrioridad

```
public static java.lang.Integer getPrioridad()
```

Devuelve el valor de prioridad

Returns:

Prioridad Valor de prioridad

---

decrementarPrioridad

```
public static void decrementarPrioridad()
```

Decrementa en uno el valor de Prioridad

---

decrementarPrioridadDoble

```
public static void decrementarPrioridadDoble()
```

Decrementa en dos el valor de la prioridad

---

aumentarESTADO

```
public static void aumentarESTADO()
```

Aumenta el valor de la variable ESTADO

---

getESTADO

```
public static java.lang.Integer getESTADO()
```

Devuelve el valor de ESTADO

Returns:
ESTADO Valor de ESTADO

---

generadorPredicciones

```
public static void generadorPredicciones(java.io.File fichReglasCLIPS,
                                         OntologiaClass Ontologia,
                                         java.io.File fichOntologiaCLIPS)
                                  throws java.lang.Exception
```

Este método se encarga de escribir todo el contenido de generador-predicciones.clp

Parameters:
`fichReglasCLIPS` - Archivo donde se va a escribir
`Ontologia` - Ontología que contiene toda la información de la BBDD
Throws:
`java.lang.Exception`

---

escribirSlotCalculoOntologia

```
public static void escribirSlotCalculoOntologia(java.io.PrintWriter fOntologia,
                                                 java.lang.String Slot)
```

Método para escribir el Slot en la Ontología Genérica

Parameters:
`fOntologia` - Fichero de la ontología genérica
`Slot` - Slot a escribir en la Ontología

Constructor Detail

TraductorRedondeo

```
public TraductorRedondeo()
```

Method Detail

traducirRedondeo

```
public static void traducirRedondeo(java.io.PrintWriter fSalida,
                                     java.io.PrintWriter fOntologia,
                                     java.lang.Integer prioridad,
                                     ReglaClass regla,
                                     OntologiaClass Ontologia)
```

Método que sirve para crear la regla de redondeo

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`prioridad` - Prioridad de la regla
`regla` - Regla que contiene el redondeo
`Ontologia` - Ontología cargada de la BBDD

Constructor Detail

TraductorDecisiones

```
public TraductorDecisiones()
```

Method Detail

traducirDecision

```
public static void traducirDecision(java.io.PrintWriter fSalida,
                                    java.io.PrintWriter fOntologia,
                                    java.lang.Integer prioridad,
                                    ReglaClass regla,
                                    OntologiaClass Ontologia)
```

Recibe una regla de tipo Decisión y escribe su traducción en el fichero generador-predicciones.clp

Parameters:

`fSalida` - Flujo de salida del fichero
`prioridad` - Valor de la prioridad
`regla` - Regla con la decisión a traducir
`Ontologia` - Ontología cargada de la BBDD

Constructor Detail

TraductorImprimir

```
public TraductorImprimir()
```

Method Detail

traducirImprimir

```
public static void traducirImprimir(java.io.PrintWriter fSalida,
                                    java.lang.Integer prioridad,
                                    java.lang.String Modulo,
                                    java.lang.String[] reglas)
```

Método que escribe la regla para imprimir los resultados

Parameters:

`fSalida` - Fichero GeneradorPredicciones a generar
`prioridad` - Prioridad de la regla
`Modulo` - Módulo en el que nos encontramos
`reglas` - Resultados de la reglas que hay que mostrar en la regla imprimir

Constructor Detail

TraductorCalculos

```
public TraductorCalculos()
```

Method Detail

traducirCalculo

```
public static void traducirCalculo(java.io.PrintWriter fSalida,
                                   java.io.PrintWriter fOntologia,
                                   ReglaClass regla,
                                   OntologiaClass Ontologia)
```

Método que traduce un cálculo a CLIPS y lo escribe en el fichero

Parameters:

`fSalida` - Fichero de salida donde se escribirá
`regla` - Regla a descomponer y traducir
`Ontologia` - Ontología cargada de la BBDD

Constructor Detail

TraductorSumatorioSimpleMercado

```
public TraductorSumatorioSimpleMercado()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de un atributo de todas las empresas del mercado

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre de la regla actual
`sumatorio` - Cálculo escrito en lenguaje Maxima del sumatorio

Constructor Detail

TraductorSumatorioSimpleSector

```
public TraductorSumatorioSimpleSector()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de un atributo de las empresas de un sector

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre de la regla actual
`sumatorio` - Cálculo escrito en lenguaje Maxima del sumatorio

Constructor Detail

TraductorSumatorioAcumuladoMercado

```
public TraductorSumatorioAcumuladoMercado()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de los 4 períodos anteriores para todo el mercado

Parameters:

`fSalida` - Fichero Generador Predicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre del cálculo actual en el que nos encontramos
`sumatorio` - Cálculo en lenguaje Maxima que contiene el sumatorio

Constructor Detail

TraductorSumatorioAcumuladoSector

```
public TraductorSumatorioAcumuladoSector()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de los 4 períodos anteriores para todo un sector

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre de la regla actual
`sumatorio` - Cálculo escrito en lenguaje Maxima del sumatorio

Constructor Detail

TraductorSumatorioEmpresasMercado

```
public TraductorSumatorioEmpresasMercado()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de todas las empresas del mercado dividido entre el nº de empresas

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre de la regla actual
`sumatorio` - Cálculo escrito en lenguaje Maxima del sumatorio

Constructor Detail

TraductorSumatorioEmpresasSector

```
public TraductorSumatorioEmpresasSector()
```

Method Detail

traductor

```
public static void traductor(java.io.PrintWriter fSalida,
                             java.io.PrintWriter fOntologia,
                             OntologiaClass Ontologia,
                             java.lang.String Modulo,
                             java.lang.Integer Prioridad,
                             java.lang.String nombre,
                             java.lang.String sumatorio)
```

Método para escribir las reglas para el sumatorio de las empresas de un sector dividido entre las empresas del sector

Parameters:

`fSalida` - Fichero GeneradorPredicciones
`fOntologia` - Fichero OntologíaGenérica
`Ontologia` - Ontología cargada de la BBDD
`Modulo` - Módulo en el que nos encontramos
`Prioridad` - Prioridad a establecer
`nombre` - Nombre de la regla actual
`sumatorio` - Cálculo escrito en lenguaje Maxima del sumatorio

10.BIBLIOGRAFÍA Y REFERENCIAS

INTRODUCCIÓN

- www.simuladoresempresariales.com
- http://e-archivo.uc3m.es/handle/10016/6825
- http://es.wikipedia.org/wiki/Sistema_basado_en_reglas
- personales.unican.es/gutierjm/cursos/expertos/Reglas.pdf
- es.wikipedia.org/wiki/CLIPS
- www.cmigestion.es/2008/gestion-empresarial/brms-business-rules-management-system/
- www.jboss.org/drools/documentation

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN

- http://www.mysql.com
- www.mamp.info/en/documentation/index.html
- clipsrules.sourceforge.net
- www.cs.princeton.edu/~appel/modern/java/JLex
- openfecks.wordpress.com/2010/01/17/tutorial-jlex-y-java-cup
- www.cs.princeton.edu/~appel/modern/java/CUP
- http://www.java.com
- www.oracle.com/technetwork/java/javase/documentation/
- netbeans.org/kb/index.html
- http://www.dragmath.bham.ac.uk
- maxima.sourceforge.net